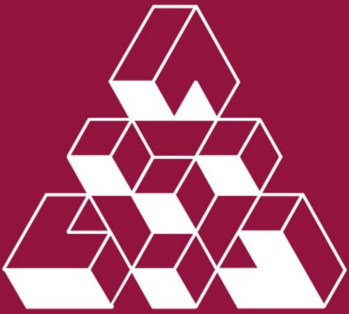
LNCS 4162

Rastislav Královič
Paweł Urzyczyn (Eds.)

Mathematical Foundations of Computer Science 2006

31st International Symposium, MFCS 2006
Stará Lesná, Slovakia, August/September 2006
Proceedings



Springer

Rastislav Královič   Paweł Urzyczyn (Eds.)

Mathematical Foundations of Computer Science 2006

31st International Symposium, MFCS 2006
Stará Lesná, Slovakia, August 28-September 1, 2006
Proceedings

Springer

Volume Editors

Rastislav Královič
Comenius University
Bratislava, Slovakia
E-mail: kralovic@dcs.fmph.uniba.sk

Paweł Urzyczyn
Warsaw University, Poland
E-mail: urzy@mimuw.edu.pl

Library of Congress Control Number: 2006930918

CR Subject Classification (1998): F.1, F.2, F.3, F.4, G.2, E.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-37791-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-37791-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11821069 06/3142 5 4 3 2 1 0

Preface

The series of Mathematical Foundations of Computer Science symposia has a well-established tradition dating back to 1972. Since the first meeting held in the Polish town of Jabłonna, the conference has gradually gained international recognition as an event bringing together researchers in all branches of theoretical computer science, promoting international cooperation, and encouraging high-quality research.

The present volume is a collection of papers presented at the 31st MFCS held in Stará Lesná, Slovakia, from August 28 to September 1, 2006. The scientific program of the 31st MFCS consisted of 62 contributed papers selected from the record high number of 174 submissions representing various areas of theoretical computer science and its mathematical foundations, complemented by 7 invited talks given by prominent researchers in the area. The symposium took place in the Academia Hotel resort situated at the foot of the Lomnický peak in the eastern part of the Vysoké Tatry Mountains in Slovakia, on the border of the Tatra National Park, close to the picturesque towns of Tatranská Lomnica and Starý Smokovec.

During its rich history, MFCS has been held in a number of places in Poland, Slovakia, and the Czech Republic, always striving to present the highest quality research in areas ranging from algorithms and data structures, to complexity, automata, semantics, logic, formal specifications, models of computation, concurrency theory, computational geometry, parallel and distributed computing, networks, bioinformatics, quantum computing, cryptography, knowledge-based systems, artificial intelligence, to mention just a few. The 2006 meeting added a new page to this history, an addition that was made possible thanks to the effort of many people.

As editors of these proceedings, we are very much indebted to all contributors to the scientific program. Our thanks are due to all authors who submitted their papers, thus showing their interest in MFCS, to all invited speakers who were willing to attend the event and share their insights, to all members of the Program Committee who did excellent work in the very difficult decision process, and to all external referees without whose help it would not be possible to evaluate so many contributions in so little time. We also gratefully acknowledge the use of the EasyChair conference system. Our thanks extend to the organizing team lead by Vanda Hambálková and Dana Pardubská, without whom this meeting could not take place. Finally, we would like to thank Springer for their professional co-operation in printing this volume, and to all participants for attending MFCS 2006.

June 2006 Rastislav Královič and Paweł Urzyczyn

Organization

Program Committee

Viviana Bono (Turin)
Ilaria Castellani (Sophia Antipolis)
Iliano Cervesato (New Orleans)
János Csirik (Szeged)
Jurek Czyzowicz (Gatineau)
Andrzej Filinski (Copenhagen)
Yuri Gurevich (Redmond WA)
Juraj Hromkovič (Zurich)
Joanna Jędrzejowicz (Gdańsk)
Juhani Karhumäki (Turku)
Rastislav Královič (Bratislava), Co-chair
Luděk Kučera (Prague)
Alberto Marchetti-Spaccamela (Rome)
Burkhard Monien (Paderborn)
Peter D. Mosses (Swansea)
Joachim Niehren (Lille)
Jaroslav Opatrný (Montreal)
José Rolim (Geneva)
Michael I. Schwartzbach (Århus)
Christian Scheideler (Munich)
Sergei Soloviev (Toulouse)
Andrzej Szepietowski (Gdańsk)
Jacobo Torán (Ulm)
Paweł Urzyczyn (Warsaw), Co-chair
Andrei Voronkov (Manchester)
Imrich Vrťo (Bratislava)
Igor Walukiewicz (Bordeaux)
Gerhard Woeginger (Eindhoven)
Shmuel Zaks (Haifa)

Organization

Vanda Hambálková
Vladimír Koutný
Edita Máčjová
Marek Nagy

The conference was organized by the Slovak Society for Computer Science and Comenius University in Bratislava.

Previous Symposia

Jabłonna, Poland, 1972
Štrbské Pleso, Czechoslovakia, 1973
Jadwisin, Poland, 1974
Mariánske Lázně, Czechoslovakia, 1975
Gdańsk, Poland, 1976
Tatranská Lomnica, Czechoslovakia, 1977
Zakopane, Poland, 1978
Olomouc, Czechoslovakia, 1979
Rydzyna, Poland, 1980
Štrbské Pleso, Czechoslovakia, 1981
Prague, Czechoslovakia, 1984
Bratislava, Czechoslovakia, 1986
Karlovy Vary, Czechoslovakia, 1988
Porąbka-Kozubnik, Poland, 1989
Banská Bystrica, Czechoslovakia, 1990
Kazimierz Dolny, Poland, 1991
Prague, Czechoslovakia, 1992
Gdańsk, Poland, 1993

Košice, Slovakia, 1994
Prague, Czech Republic, 1995
Kraków, Poland, 1996
Bratislava, Slovakia, 1997
Brno, Czech Republic, 1998
Szklarska Poręba, Poland, 1999
Bratislava, Slovakia, 2000
Mariánske Lázně, Czech Republic, 2001
Warsaw, Poland, 2002
Bratislava, Slovakia, 2003
Prague, Czech Republic, 2004
Gdańsk, Poland, 2005

Referees

Eric Allender
Luca Allulli
Giuseppe Ateniese
Cedric Bastien
Tugkan Batu
Marek Bednarczyk
Giuseppe Berio
Simona Bernardi
Luca Bernardinello
Dietmar Berwanger
Sergei Bezrukov
Vittorio Bilò
Manuel Bodirsky
Mikołaj Bojańczyk
Vincenzo Bonifaci
Sem Borst
Béatrice Bouchou
Claus Brabrand
Franck van Breugel
Gerth Brodal
Anne Brüggemann-Klein
Andrei Bulatov
Peter Bürgisser
Hal Burch
Nadia Busi
Hans-Joachim Böckenhauer
Tiziana Calamoneri
Anne-Cécile Caron
Dario Catalano
Bogdan Chlebus
Piotr Chrząstowski
Andrea Clementi
Eugen Czeizler
Flavio D'Alessandro
Ivan Damgård
Ferruccio Damiani
Carsten Damm
Robert Dąbrowski
Rocco De Nicola
Olivier Devillers
Srikrishnan Divakaran
Stefan Dobrev
Debora Donato
Arnaud Durand
Jean-Louis Durieux
Pavol Ďuriš
Roy Dyckhoff
Stefan Dziembowski
Robert Elsaesser
Zoltán Ésik
Piotr Faliszewski
Angelo Fanelli
Tomas Feder
Uriel Feige
Rainer Feldmann
Louis Feraud
Guillaume Fertin
Barbara Fila
Philippe Flajolet
Rudolf Fleischer
Michal Forišek
Enrico Formenti
Lance Fortnow
Wit Foryś
Gudmund S. Frandsen
Martin Gairing
William Gasarch
Marie-Claude Gaudel
Fanica Gavril
Dan Geiger
Blaise Genest
Hugo Gimbert
Françoise Gire
Christian Glaßer
Anna Gomolińska
Paweł Górecki
Chris Gray
Sven Grothklags
Dan Gutfreund
Vesa Halava
Carmem Hara
Tero Harju
Nick Harvey
Herman Haverkort
Pinar Heggernes
Keijo Heljanko
Miki Hermann
Andreas Herzig
Daniel Hirschkoff
Tom Hirschowitz
Mika Hirvensalo
John Hitchcock
Piotr Hoffman
Juha Honkala
Karol Horodecki
Peter Høyer
Paweł Idziak
Lucian Ilie
Kazuo Iwama
Matthias Jantzen
Aubin Jarry

Alan Jeffrey
Markus Junker
Michael Kaminski
Jarkko Kari
Tomi Karki
Marek Karpinski
Branislav Katreniak
Dan Kenigsberg
Peter G. Kimmel
Christian Kirkegaard
Ralf Klasing
Bartek Klin
Martin Kochol
Łukasz Kowalik
Mirosław Kowaluk
Richard Kralovič
Dieter Kratsch
Andrei Krokhin
Piotr Krzyżanowski
Grégory Kucherov
Narayan Kumar
Joachim Kupke
Giovanni Lagorio
Sławomir Lasota
Aurélien Lemay
Stéphane Lengrand
Pierre Leone
Arto Lepisto
Jerome Leroux
Peter Leupold
Asaf Levin
Stefan Lietsch
Maciej Liśkiewicz
Satyarayana V. Lokam
Sylvian Lombardy
Ulf Lorenz
Christof Löding
Jack Lutz
Olivier Ly
Daniel Marx
Ralph Matthes
Marios Mavronicolas
Daniel Meister
Michael Mislove
Angelo Montanari
Tal Mor
Luminita Moraru
Luca Moscardelli
Philippe Moser
Andrzej W. Mostowski
Marian Mrozek
Makoto Murata
Anca Muscholl
Tobias Mömke
Anders Møller
Boaz Nadler
Alfredo Navarra
Roman Nedela
Calvin Newport
Hung Son Nguyen
Rolf Niedermeier
Edward Ochmański
Enno Ohlebusch
Alexander Okhotin
Nicola Olivetti
Catuscia Palamidessi
Dana Pardubská
Paweł Pączkowski
Marcin Peczarski
Rudi Pendavingh
Carla Piazza
Rom Pinchasi
Wojciech Plandowski
Leszek Plaskota
Piotr Pokarowski
Olivier Powell
Gian Luca Pozzato
Gabriele Puppis
Danny Raz
Maxime Rebout
Wolfgang Reisig
Renato Renner
Andrea Ribichini
Éric Rivals
Antoine Rollet
Yves Roos
Guenter Rote
Jörg Rothe
Andrzej Ruciński
Wojciech Rytter
Kalle Saari
Kai Salomaa
Piotr Sankowski
Nicolae Santean
Thomas Sauerwald
Marcus Schaefer
Stefan Schamberger
Peter Schneider-Kamp
Florian Schoppmann
Ulf-Peter Schroeder
Christoph Schwarzweller
Sebastian Seibert
Olivier Serre
Peter Sewell
Géraud Sénizergues
Jiří Sgall
Hadas Shachnai
Farhad Shahrokhi
Andrea Silvestri
Jens Simon
Mitali Singh
Christian Sohler
Paul Spirakis
Jeremy Sproston
Ladislav Stacho
Martin Strecker
Michał Strojnowski
Madhu Sudan
Maxim Sviridenko
Marcin Szczuka
Błażej Szepietowski
Siamak Taati
Jean-Marc Talbot
Patrizia Tavella
P.S. Thiagarajan
Karsten Tiemann
Sophie Tison
Fabien Torre
Géza Tóth
Stephen Travers
Tobias Tscheuschner
Jerzy Tyszkiewicz
Tomasz Urbański
Jorge Urrutia
Frits Vaandrager

György Vaszil	Klaus Wagner	Mordechai Shalom
Venkat Venkateswaran	Uli Wagner	Hans Zantema
Annamaria Vernone	Charles R. Wallace	Zhenjie Zhang
Maria-Grazia Vigliotti	Pascal Weil	Andrei Zinovyev
Andrea Vitaletti	Michael Weiss	Alex Znamenshchykov
Tjark Vredeveld	Duminda Wijesekera	

Table of Contents

A Core Calculus for Scala Type Checking

Vincent Cremet[1], François Garillot[2], Sergueï Lenglet[3], and Martin Odersky[1]

[1] École Polytechnique Fédérale de Lausanne
INR Ecublens, 1015 Lausanne, Switzerland
[2] École Normale Supérieure
45 rue d'Ulm, 75230 Paris, France
[3] École Normale Supérieure de Lyon
46 alle d'Italie, 69364 Lyon, France

Abstract. We present a minimal core calculus that captures interesting constructs of the Scala programming language: nested classes, abstract types, mixin composition, and path dependent types. We show that the problems of type assignment and subtyping in this calculus are decidable.

1 Introduction

The programming language Scala proposes a new model for component systems [28]. Components in this model are classes, which can be combined using nesting and mixin composition. Classes can contain abstract types which may be instantiated in subclasses. The Scala component model thus provides a single framework for the construction of objects and modules. Modules are identified with objects, functors with classes, and signatures with traits.

The advantage of this approach is that a single fairly small set of language constructs is sufficient for core programming as well as the definition of components and their composition. Furthermore, the identification of modules and objects provides new ways to formulate standard programming tasks such as the expression problem [14,33,27] and family polymorphism [12,28].

Scala's approach to component modeling is based on three programming language constructs: modular mixin composition, abstract type members, and explicit self-types. All three have been studied in the νObj calculus [25]. A key concept of the νObj calculus, path-dependent types, is also present in Scala. However, some other constructions of νObj do not correspond to Scala language constructs. In particular, νObj has first-class classes which can be passed around as values, but Scala has not.

First-class classes were essential in establishing an encoding of $F_{<:}$ in νObj, which led to a proof of undecidability of νObj by reduction to the same property in $F_{<:}$ [29]. However, since Scala lacks first-class classes, the undecidability result for the calculus does not imply that type checking for the programming language is undecidable.

In this paper, we study the problem of decidability of Scala type checking. We construct (algorithmic) Featherweight Scala, abbreviated FS_{alg}, a minimal core calculus of classes that captures an essential set of features of Scala's type system.

Classes can have types, values, methods and other classes as members. Types, methods, and values can be abstract. The calculus is designed to be syntactically a subset of Scala (with the deviation of explicit self-names, explained below). Its typing rules correspond closely to the ones implemented in the Scala compiler.

An important aim in developping Featherweight Scala was to show that Scala's core type-checking rules are decidable. One particular problem in this respect are cyclic definitions that relate members of different classes. Featherweight Scala allows cyclic references between members of different mixin classes. Because cycles cannot be ruled out by construction, they have to be detected by the type checker. The typing rules achieve this by keeping track of the sets of definitions that have already been visited in a typing proof. This gives the calculus an algorithmic flavor, hence the name FS_{alg}.

All presented deduction rules are syntax-directed and thus lead directly to procedures for subtyping and type assignment. The central result of this paper is that these procedures are algorithms, i.e. that they terminate in each case.

Related Work: Scala's component constructions provide a middle ground between the worlds of object-oriented programming and functional module systems [15,21]. Many of the concepts in both worlds are unified. Mixin composition in Scala borrows from mixin-modules [3,4,16], as well as from the more linear object-oriented mixin composition [6,5]. Components in Scala can be mutually recursive, a property which is also addressed by work on recursive modules [9,23]. Abstract types in Scala are also present in SML style signatures [21,15], and correspond almost exactly to virtual classes in Beta [22].

Later work on virtual classes [11,13] is more general in that classes, and not just types, can be abstract. However, references to abstract types in [13] can only refer to members of an enclosing "self", not to members of an arbitrary path. Several other variations on virtual types [32,17,24] and some alternative proposals [8,31,7,20] have also been researched. Typed class-based calculi for describing Scala's static analysis are described in [2] and [1] but the authors do not address the problem of their decidability.

The focus in the paper is on a minimal calculus that captures the essential features of an existing programming language. In this motivation it follows the work on Featherweight Java [19]. Both calculi model a simple, purely functional core language of objects with fields and methods in a nominal class-based type system. They also take some similar shortcuts in the interest of conciseness. For instance, both assume call-by-name evaluation in order not to have to deal with the thorny initialization issues of their underlying languages. However, the set of more advanced language constructs that are modeled are different in each case. Featherweight Java models type casts, and has extensions that model generics as well as inner classes [18]. Featherweight Scala models inner classes, member type abstraction, as well as path-dependent types. It can also model most of the generic constructs in FGJ [19] via encodings.

The rest of this paper is structured as follows. Section 2 explains Scala's model of abstract types and path dependent types from a programmer's perspective. Section 3 introduces the Featherweight Scala calculus FS_{alg}. Section 4 shows that

```
trait Any extends { this0 | }
trait Nat extends Any { this0 |
    def isZero(): Boolean
    def pred(): Nat
    trait Succ extends Nat { this1 |
        def isZero(): Boolean = false
        def pred: Nat = this0
    }
    def succ(): Nat = { val result = new Succ {}; result }
    def add(other: Nat): Nat = {
        if (this0.isZero()) other else this0.pred.add(other.succ())
    }
    def subtract(other: Nat): Nat = {
        if (other.isZero()) this0 else this0.succ().subtract(other.pred)
    }
}

val zero = new Nat { this0 |
    def isZero(): Boolean = true
    def pred: Nat = error("zero.pred")
}
```

Fig. 1. Definition of Peano numbers

subtyping and type-assignment are decidable in this calculus. Section 5 relates the obtained results to the situation in the Scala language. Section 6 concludes.

2 Programming in Featherweight Scala

Featherweight Scala is a fairly small subset of Scala, but it is expressive enough for one to write meaningful programs in it. In the following, we show how some common classes and programming idioms can be encoded in the calculus.

Peano Numbers. We start with an encoding of Peano numbers, shown in Figure 1. This encoding presents a trait *Nat* with five member methods. Methods *isZero* and *pred* are *abstract*, that is, they lack an implementation in trait *Nat*. By contrast, methods *succ*, *add* and *subtract* are *concrete*. A *trait* in Scala is an abstract class which may be combined with other traits using mixin composition.

References from one member of a *Nat* object to another always go via the "self" reference of the class. The name of the self-reference is given after the opening brace of the class body. In the example above it is *this0*. The name can be freely chosen, but in the examples in this paper we always use *thisN*, where *N* is the nesting level of the enclosing class. As a shorthand notation we sometimes omit the definition of a self-name which is never used in the class body that follows.

Class *Nat* also contains a nested class *Succ* which defines the successor value of the current object *this0*. This class is an extension of *Nat*, which gives concrete definitions for the two abstract members of *Nat*: *isZero* returns always ***false*** and the predecessor method *pred* always returns the self-reference of the outer *Nat* class.

The successor method *succ* simply creates a new instance of the *Succ* class and returns it. Note that Featherweight Scala allows instance creation expressions such as ***new*** *Succ* {} only as right-hand sides of value definitions; that's why we were forced to define in the body of *succ* an intermediate value *result*. Regular Scala does not have this restriction, and also provides many other shorthands that would make the example more pleasant to write and read.

The final two methods, *add* and *subtract*, define addition and subtraction in terms of the previous three methods. Their implementation uses standard syntax for field selection, method calls, and recursion.

Figure 1 also gives a definition of the *zero* value for Peano numbers. This value is defined as a direct specialization of class *Nat*, which also defines the natural implementations of *Nat*'s abstract methods *isZero* and *pred*. The right-hand side of *zero*'s definition combines a definition of a new anonymous class and a creation of an instance of this class in a single syntactic construct. Alternatively, one could also proceed in two steps, by defining a subclass *Zero* of *Nat*, and then creating an instance value ***val*** *zero* = ***new*** *Zero* {}.

The preceding example used exclusively the constructs of the Featherweight Scala calculus, with two exceptions: First, we assumed a type *Boolean* with values ***true*** and ***false*** and a *if-then-else* construct. Second, we assumed an *error* function which aborts a program with a given error message.

In the example we have also taken some liberty in presenting top-level definitions for *Nat* and *zero*. By contrast, a Featherweight Scala program is simply an expression, which typically contains embedded definitions for classes and values. To get a complete program which computes *2+2* one could combine the program fragments in Figure 1 as follows:

```
val universe = new { global |
    class Nat { ... }
}
val zero = new universe.Nat { ... };
val two = zero.succ().succ();
two.add(two)
```

In this program, references from one top-level definition in *universe* to another would go via the top-level self-reference *global*.

The syntax used in the examples in this section is also regular Scala, with one exception: Scala does not have a clause { *thisN* | ... } which names the self-reference *thisN* of the enclosing class. Instead, one always uses the reserved word ***this***. Self-references of an outer class *C* can be denoted by prefixing ***this*** with the name of the outer class, e.g., *C.**this***.

Lists. As a second example, Figure 2 presents a class hierarchy for lists in Featherweight Scala. There are three classes: A base class *List* and two subclasses *Cons* and *Nil* that define non-empty and empty lists, respectively.

```
trait List extends Any { this0 |
    type Elem
    type ListOfElem = List { this1 | type Elem = this0.Elem }
    def isEmpty(): Boolean
    def head(): this0.Elem
    def tail(): this0.ListOfElem
}
trait Nil extends List { this0 |
    def isEmpty(): Boolean = true
    def head(): this0.Elem = error("Nil.head")
    def tail(): this0.ListOfElem = error("Nil.tail")
}
trait Cons extends List { this0 |
    val hd: this0.Elem
    val tl: this0.ListOfElem
    def isEmpty(): Boolean = false
    def head(): this0.Elem = hd
    def tail(): this0.ListOfElem = tl
}
```

Fig. 2. Definition of the List class hierarchy

Lists can have arbitrary element types. In the standard Scala library, this is expressed by a parameterized type, but the featherweight version does not have type parameters. Instead, we use *abstract types* to express the genericity of the list abstraction.

The element type of a given list is represented by the type member *Elem* of class *List*. The member is defined as an abstract type in class *List*. Hence, when a *List* object is created, a concrete implementation of this type has to be provided. For instance, the definition

val *nilOfNat* = ***new*** *Nil* { ***type*** *Elem* = *Nat* }

defines a value *nilOfNat* as an empty list with element type *Nat*. The *type alias* ***type*** *Elem* = *Nat* is used to "fill in" the abstract type member *Elem* that *Nil* inherits from *List*.

The *List* class also contains a type alias which defines *ListOfElem* as a type name for lists whose element type is the same as the element type of the list in question. This alias does not implement an abstract type member in a parent class; it is there only for convenience.

The *List* class also defines a test method *isEmpty*, as well as methods which return the head and tail of a list. Method *head* returns values of type *Elem* whereas *tail* returns values of type *ListOfElem*. All three methods are abstract in class List.

The subclass *Nil* of *List* represents empty lists. It defines method *isEmpty* to return ***true***. Selecting the head or tail of an empty list always results in an error.

The subclass *Cons* of *List* represents nonempty lists. The head and tail of a non-empty list are kept in the fields *hd* and *tl* of class *Cons*. Scala uses ***val*** for a definition of a local value or a field of a class, whereas ***def*** is used for a method definition. Classes in Featherweight Scala do not have constructors; however, one can use member-redefinition to initialize the values of an object. For instance, the following code defines two lists of element type *Nat* which contain the values (*2*) and (*1, 2*), respectively.

```
val list2 = new Cons { this0 |
    type Elem = Nat
    val hd: Nat = zero.succ().succ()
    val tl: this0.ListOfElem = nilOfNat
}
val list12 = new Cons { this0 |
    type Elem = Nat
    val hd: Nat = zero.succ()
    val tl: this0.ListOfElem = list2
}
```

The *List* example showed how genericity can be encoded using abstract types. In fact, there is a general encoding that lets one encode all forms of parameterized types in Scala into types with abstract members. Details are found in [1].

```
trait Function extends Any { this0 |
    type Dom
    type Range
    def apply(x: Dom): Range
}
val inc = new Function { this0 |
    type Dom = Nat
    type Range = Nat
    def apply(x: this0.Dom): this0.Range = x.succ()
}
```

Fig. 3. Definition of first-class functions

Higher-Order Functions. Featherweight Scala has methods, i.e. function-valued class-members, but it has no function-valued parameters or results. However, it is possible to encode first-class functional values as instances of a standard *Function* class, which is presented in Figure 3. The current Scala implementation uses a similar encoding to map functional values to the JVM.

Class *Function* gives an interface for functions with arbitrary domain and range types. The interface specifies two abstract types *Dom* and *Range* as well as an method abstract method *apply* that takes arguments of type *Dom* and that yields results of type *Range*.

A first-class functional value is then a concrete implementation of class *Function*, which gives types for *Dom* and *Range* as well as an implementation for method *apply*. Figure 3 shows as an example a first-class incrementer function *inc* over Peano numbers.

```
trait Mapper extends Any { this0 |
    type A
    type B
    def map(f: Function { type Dom = this0.A; type Range = this0.B },
            xs: List { type Elem = this0.A }): List { type Elem = this0.B } =
        if (xs.isEmpty()) {
            val result = new Nil {
                type Elem = this0.B
            };
            result
        } else {
            val result = new Cons {
                type Elem = this0.B
                val hd = f.apply(xs.head())
                val tl = this0.map(f, xs.tail())
            };
            result
        }
}
```

Fig. 4. Encoding of the higher-order *map* function

Since first-class functions are objects, they can be passed around like any other value. To apply as first-class function, one simply invokes its apply method (regular Scala defines syntactic sugar so this is done automatically whenever a first-class function value appears in function position in an application).

As an example, Figure 4 presents a *map* function which applies a given argument function to all elements of a given list and returns a list consisting of all the results of these applications. In regular Scala, this function would be defined as follows:

```
def map[A, B](f: A ⇒ B, xs: List[A]): List[B] =
    if (xs.isEmpty) Nil else f(x.head) :: map(f, xs.tail)
```

Since *map* is conceptually a polymorphic method, its encoding in Featherweight Scala makes use of a wrapper class *Mapper* which defines two abstract types A and B, representing the element types of the argument and result lists, respectively.

The *map* method in *Mapper* takes as arguments a function f from type A to type B, and a list *xs* of element type A. It returns a list of element type B. An application of *map* would be written as follows:

```
val list23 = new Mapper { type A = Nat; type B = Nat }.map(inc, list12)
```

This instantiates the *Mapper* class with type *Nat* as the element type of the argument and result list, and invokes the *map* method of the instantiation with *inc* and *list12* as arguments. The expression would return the encoding of the list (*2*, *3*).

The example shows that monomorphic functions such as *inc* can be first-class values. However, the construction cannot be generalized to polymorphic functions. The reason is that polymorphic functions like *map* have to be encoded using wrapper classes. Such wrapper classes are first-class values neither in Featherweight nor in regular Scala. By contrast, the νObj calculus has classes as first class values, and therefore can encode polymorphic functional values.

3 The Algorithmic Featherweight Scala Calculus

The FS_{alg} calculus aims at describing some central aspects of the Scala type system in a simple and formal way. The features whose study has been privileged in this work are: method overriding, mixins, inner classes, virtual types, singleton types and types with member refinements. The calculus does not model objects with state and has no concept of type parameters in classes or methods. Genericity can be encoded using types with member refinements.

3.1 Syntax

The abstract syntax of FS_{alg} is given in Figure 5. Amongst the names occurring in a program, we distinguish the *variables* that are used as binders for objects and can be α-conversed, the *value labels* that designate the members defining a field or a method, and the *type labels* that designate the members defining a class or a virtual type.

A *member* can be a value field, a method, a type field or a class. A value field is immutable and refers to an object. The types of value fields, method parameters and method results must be given explicitly. Value fields and methods can be either concrete or abstract. A declaration is called abstract if the right-hand side is absent. For instance, a declaration like $\texttt{val}_n a : T = t$ defines a concrete field a with value t, whereas the field declaration $\texttt{val}_n a : T$ is abstract. In order to factorize abstract and concrete declarations we use the notation $\texttt{val}_n a : T\,(= t)^?$. Type fields are also either concrete or abstract, a concrete type field being sometimes called a *type alias*. A class member $\texttt{trait}_n A\,\texttt{extends}\,\left(\overline{T}\right)\,\left\{\varphi \,|\, \overline{M}\right\}$ declares a class A with parents $\overline{T}$ and members $\overline{M}$, the variable φ denotes the current instance of the class. Class members cannot be abstract.

Every occurrence of a declaration in a program is tagged with a unique integer n. This integer has no computational meaning, it is simply used for detecting cycles during the static analysis.

The *terms* of the calculus are standard. A variable x can represent the current instance of a class, a method parameter, or the name of an object which has been created locally. Field selections and method calls have the same syntax as in Java. The construct $\texttt{val}\ x = \texttt{new}\ T; t$ allows the user to define a new instance of the type T, with a name x whose scope is limited to the term t.

A FS_{alg} *program* is simply a term, which is usually of the form

$$\texttt{val}\ z = \texttt{new}\ \left\{\varphi \,|\, \overline{M}\right\}; t\ .$$

It consists of a list of member declarations $\overline{M}$ that together make up a *universe* object z and a main term t to be evaluated in the context of z. The variable

Syntax

x, y, z, φ	Variable
a	Value label
A	Type label
$P ::=$	Program
$\{x \mid \overline{M}\ t\}$	
$M, N ::=$	Member decl
$\mathtt{val}_n a : T\ (= t)^?$	Field decl
$\mathtt{def}_n a\ \left(\overline{y : S}\right) : T\ (= t)^?$	Method decl
$\mathtt{type}_n A\ (= T)^?$	Type decl
$\mathtt{trait}_n A\ \mathtt{extends}\ \left(\overline{T}\right) \{\varphi \mid \overline{M}\}$	Class decl
$s, t, u ::=$	Term
x	Variable
$t.a$	Field selection
$s.a\ \left(\overline{t}\right)$	Method call
$\mathtt{val}\ \ x = \mathtt{new}\ T; t$	Object creation

$p ::=$	Path
x	Variable
$p.a$	Field selection
$S, T, U ::=$	Type
$p.A$	Type selection
$p.\mathbf{type}$	Singleton type
$\left(\overline{T}\right) \{\varphi \mid \overline{M}\}$	Type signature

Reduction

$$\frac{\mathtt{val}_n a : T = t \in \Sigma(x)}{\Sigma\ ;\ x.a \rightarrow \Sigma\ ;\ t} \quad (\textsc{red-value})$$

$$\frac{\mathtt{def}_n a\ \left(\overline{z : S}\right) : T = t \in \Sigma(x)}{\Sigma\ ;\ x.a(\overline{y}) \rightarrow \Sigma\ ;\ [\overline{y}/\overline{z}]t} \quad (\textsc{red-method})$$

$$\frac{\Sigma \vdash T \prec_x \overline{M}}{\Sigma\ ;\ \mathtt{val}\ \ x = \mathtt{new}\ T; t \rightarrow \Sigma, x : \overline{M}\ ;\ t} \quad (\textsc{red-new})$$

$$\frac{\Sigma\ ;\ t \rightarrow \Sigma'\ ;\ t'}{\Sigma\ ;\ e[t] \rightarrow \Sigma'\ ;\ e[t']} \quad (\textsc{red-context})$$

Lookup

$$\frac{\forall i,\ \Sigma \vdash T_i \prec_\varphi \overline{N_i}}{\Sigma \vdash \left(\overline{T}\right) \{\varphi \mid \overline{M}\} \prec_\varphi \left(\biguplus_i \overline{N_i}\right) \uplus \overline{M}} \quad (\textsc{lookup-sig})$$

$$\frac{\begin{array}{c}\mathtt{trait}_n A\ \mathtt{extends}\ \left(\overline{T}\right) \{\varphi \mid \overline{M}\} \in \Sigma(y) \\ \Sigma \vdash \left(\overline{T}\right) \{\varphi \mid \overline{M}\} \prec_\varphi \overline{N}\end{array}}{\Sigma \vdash y.A \prec_\varphi \overline{N}} \quad (\textsc{lookup-class})$$

$$\frac{\begin{array}{c}\mathtt{type}_n A = T \in \Sigma(y) \\ \Sigma \vdash T \prec_\varphi \overline{M}\end{array}}{\Sigma \vdash y.A \prec_\varphi \overline{M}} \quad (\textsc{lookup-alias})$$

where

$e\ ::=$ **(term evaluation context)**
- $\langle\rangle$
- $e.a$
- $e.a\ (t)$
- $x.a\ (\overline{s}, e, \overline{u})$
- $\mathtt{val}\ \ x = \mathtt{new}\ E; t$

$E ::=$ **(type evaluation context)**
- $e.A$
- $\left(\overline{T}, E, \overline{U}\right) \{\varphi \mid \overline{M}\}$

Fig. 5. The FS_{alg} Calculus : Syntax & Reduction

Path Typing

$$\frac{x : T \in \Gamma}{\mathcal{S}, \Gamma \vdash_{path} x : T}\ (\textsc{path-var}) \qquad \frac{\mathcal{S}, \Gamma \vdash p.\mathbf{type} \ni \mathtt{val}_n a : T \,(= t)^?}{\mathcal{S}, \Gamma \vdash_{path} p.a : T}\ (\textsc{path-select})$$

Type Assignment

$$\frac{\mathcal{S}, \Gamma \vdash_{path} p : T}{\mathcal{S}, \Gamma \vdash p : p.\mathbf{type}}\ (\textsc{path}) \qquad \frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash s : S \\ \mathcal{S}, \Gamma \vdash \bar{t} : \overline{T'} \qquad \mathcal{S}, \Gamma \vdash \overline{T'} <: \overline{T} \\ \mathcal{S}, \Gamma \vdash S \ni \mathtt{def}_n a \left(\overline{x : T}\right) : U \,(= u)^?\end{array}}{\mathcal{S}, \Gamma \vdash s.a\left(\bar{t}\right) : U}\ (\textsc{method})$$

$$\frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash t : S \qquad t \text{ is not a path} \\ \mathcal{S}, \Gamma \vdash S \ni \mathtt{val}_n a : T \,(= u)^?\end{array}}{\mathcal{S}, \Gamma \vdash t.a : T}\ (\textsc{select}) \qquad \frac{\begin{array}{c}\mathcal{S}, \Gamma, x : T \vdash t : S \qquad x \notin \text{fn}(S) \\ \mathcal{S}, \Gamma \vdash T \prec_\varphi \overline{M_c} \qquad \mathcal{S}, \Gamma \vdash T \ \textsc{wf}\end{array}}{\mathcal{S}, \Gamma \vdash \mathtt{val}\ \ x = \mathtt{new}\ T; t : S}\ (\textsc{new})$$

Expansion

$$\frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash p.\mathbf{type} \ni \mathtt{trait}_n A \,\mathtt{extends}\left(\overline{T}\right)\left\{\varphi \,|\, \overline{M}\right\} \\ \{n\} \cup \mathcal{S}, \Gamma \vdash \left(\overline{T}\right)\left\{\varphi \,|\, \overline{M}\right\} \prec_\varphi \overline{N} \qquad n \notin \mathcal{S}\end{array}}{\mathcal{S}, \Gamma \vdash p.A \prec_\varphi \overline{N}}\ (\prec\textsc{-class}) \qquad \frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash p.\mathbf{type} \ni \mathtt{type}_n A = T \\ \{n\} \cup \mathcal{S}, \Gamma \vdash T \prec_\varphi \overline{M} \qquad n \notin \mathcal{S}\end{array}}{\mathcal{S}, \Gamma \vdash p.A \prec_\varphi \overline{M}}\ (\prec\textsc{-type})$$

$$\frac{\forall\, i,\ \mathcal{S}, \Gamma \vdash T_i \prec_\varphi \overline{N_i}}{\mathcal{S}, \Gamma \vdash \left(\overline{T}\right)\left\{\varphi \,|\, \overline{M}\right\} \prec_\varphi \left(\biguplus_i \overline{N_i}\right) \uplus \overline{M}}\ (\prec\textsc{-signature})$$

Membership

$$\frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash p \simeq q \qquad \mathcal{S}, \Gamma \vdash_{path} q : T \\ \psi(p) \cup \mathcal{S}, \Gamma \vdash T \prec_\varphi \overline{M} \qquad \psi(p) \nsubseteq \mathcal{S}\end{array}}{\mathcal{S}, \Gamma \vdash p.\mathbf{type} \ni [p/\varphi]M_i}\ (\ni\textsc{-singleton}) \qquad \frac{\begin{array}{c}T \text{ is not a singleton type} \\ \mathcal{S}, \Gamma \vdash T \prec_\varphi \overline{M} \qquad \varphi \notin \text{fn}(M_i)\end{array}}{\mathcal{S}, \Gamma \vdash T \ni M_i}\ (\ni\textsc{-other})$$

Fig. 6. The FS_{alg} Calculus : Type Assignment, Expansion & Membership

φ is an alias of z; it represents the self reference of the universe object which contains all top-level declarations.

We distinguish a subcategory of terms that can be used inside types, and that we call *paths*. A path is either a variable or the selection of a field on a term that is itself a path. The introduction of paths is motivated by their property of always evaluating to the same object value and of being strongly normalizable. Both properties are needed if we want type soundness to hold. However, this goes beyond the scope of the present paper.

Our calculus has a rich syntax of *types*. A type selection $p.A$ is either a *class type* if A is a class label, a *virtual type* if A is an abstract type label, or an *alias*

Well-Formedness

$$\frac{\begin{array}{c}\mathcal{S},\Gamma \vdash_{path} p : T \qquad \psi(p) \not\subseteq \mathcal{S}\\ \psi(p)\cup\mathcal{S},\Gamma \vdash T \ \textsc{wf}\end{array}}{\mathcal{S},\Gamma \vdash p.\mathbf{type}\ \textsc{wf}}\ (\textsc{wf-singleton})$$

$$\frac{\mathcal{S},\Gamma,\varphi : (\overline{T})\,\{\varphi\,|\,\overline{M}\} \vdash (\overline{T})\,\{\varphi\,|\,\overline{M}\}\ \textsc{wf}_{\varphi}}{\mathcal{S},\Gamma \vdash (\overline{T})\,\{\varphi\,|\,\overline{M}\}\ \textsc{wf}}\ (\textsc{wf-signature})$$

$$\frac{\mathcal{S},\Gamma \vdash p.\mathbf{type} \ni \mathtt{trait}_n A\,\mathtt{extends}\,(\overline{T})\,\{\varphi\,|\,\overline{M}\}}{\mathcal{S},\Gamma \vdash p.A\ \textsc{wf}}\ (\textsc{wf-class})$$

$$\frac{\begin{array}{c}\mathcal{S},\Gamma \vdash p.\mathbf{type} \ni \mathtt{type}_n A\,(=T)^{?}\\ \left(\{n\}\cup\mathcal{S},\Gamma \vdash T\ \textsc{wf} \qquad n \notin \mathcal{S}\right)^{?}\end{array}}{\mathcal{S},\Gamma \vdash p.A\ \textsc{wf}}\ (\textsc{wf-type})$$

MemberWell-Formedness

$$\frac{(\mathcal{S},\Gamma \vdash T\ \textsc{wf})^{?}}{\mathcal{S},\Gamma \vdash \mathtt{type}_n A\,(=T)^{?}\ \textsc{wf}_x}\ (\textsc{wf-x-type})$$

$$\frac{\mathcal{S},\Gamma,\varphi : x.A \vdash (\overline{T})\,\{\varphi\,|\,\overline{M}\}\ \textsc{wf}_{\varphi}}{\mathcal{S},\Gamma \vdash \mathtt{trait}_n A\,\mathtt{extends}\,(\overline{T})\,\{\varphi\,|\,\overline{M}\}\ \textsc{wf}_x}\ (\textsc{wf-x-class})$$

$$\frac{\begin{array}{c}\mathcal{S},\Gamma \vdash T\ \textsc{wf}\\ (\mathcal{S},\Gamma \vdash t : T' \qquad \mathcal{S},\Gamma \vdash T' <: T)^{?}\end{array}}{\mathcal{S},\Gamma \vdash \mathtt{val}_n a : T\,(=t)^{?}\ \textsc{wf}_x}\ (\textsc{wf-x-field})$$

$$\frac{\begin{array}{c}\mathcal{S},\Gamma \vdash \overline{S}, T\ \textsc{wf}\\ \overline{S}\text{ does not contain singleton types}\\ (\mathcal{S},\Gamma,\overline{x : S} \vdash t : T' \qquad \mathcal{S},\Gamma \vdash T' <: T)^{?}\end{array}}{\mathcal{S},\Gamma \vdash \mathtt{def}_n a\,\left(\overline{x : S}\right) : T\,(=t)^{?}\ \textsc{wf}_x}\ (\textsc{wf-x-method})$$

$$\frac{\begin{array}{c}\forall i,\ \mathcal{S},\Gamma \vdash T_i \prec_{\varphi} \overline{N_i} \qquad \mathcal{S},\Gamma \vdash \overline{M}\ \textsc{wf}_{\varphi}\\ \mathcal{S},\Gamma \vdash \overline{T}\ \textsc{wf} \qquad \forall\ (i,j),\ \mathcal{S},\Gamma \vdash (\overline{N_{i+j}}, \overline{M}) \ll \overline{N_i}\end{array}}{\mathcal{S},\Gamma \vdash (\overline{T})\,\{\varphi\,|\,\overline{M}\}\ \textsc{wf}_{\varphi}}\ (\textsc{wf-x-signature})$$

Path Alias Expansion

$$\frac{\begin{array}{c}\mathcal{S},\Gamma \vdash_{path} p : q.\mathbf{type}\\ \psi(p)\cup\mathcal{S},\Gamma \vdash q \simeq q' \qquad \psi(p) \not\subseteq \mathcal{S}\end{array}}{\mathcal{S},\Gamma \vdash p \simeq q'}\ (\simeq\text{-}\textsc{Step})$$

$$\frac{\begin{array}{c}\mathcal{S},\Gamma \vdash_{path} p : T\\ T\text{ is not a singleton type}\end{array}}{\mathcal{S},\Gamma \vdash p \simeq p}\ (\simeq\text{-}\textsc{Refl})$$

Fig. 7. The FS_{alg} Calculus : Well-Formedness and Path Alias Expansion

type of A is a concrete type label. A class type $p.A$ has as values all instances of class A whose enclosing instance associated with A is the object denoted by p. Virtual and alias types $p.A$ have a rather different meaning: they represent the type held by the type field A in the object p. A singleton type $p.\mathbf{type}$ represents the type of which p is the unique element. Finally, a type signature $(\overline{T})\,\{\varphi\,|\,\overline{M}\}$ combines the concepts of intersection types and member refinements: it represents the intersection of types $\overline{T}$ with additional constraints on members expressed by declarations $\overline{M}$.

3.2 Operational Semantics

Figure 5 contains the inference rules that define a small-step operational semantics for our calculus. It is composed of a *reduction* relation and a *lookup* relation.

Type Alias Expansion

$$\frac{\mathcal{S},\Gamma \vdash p.\mathbf{type} \ni \mathtt{type}_n A = T \qquad \{n\}\cup\mathcal{S},\Gamma \vdash T \simeq U \qquad n \notin \mathcal{S}}{\mathcal{S},\Gamma \vdash p.A \simeq U}\ (\simeq\text{-TYPE})$$

$$\frac{\mathcal{S},\Gamma \vdash p.\mathbf{type} \ni \mathtt{trait}_n A\, \mathtt{extends}\ (\overline{T})\ \{\varphi \,|\, \overline{M}\}}{\mathcal{S},\Gamma \vdash p.A \simeq p.A}\ (\simeq\text{-CLASS})$$

$$\mathcal{S},\Gamma \vdash (\overline{T})\ \{\varphi \,|\, \overline{M}\} \simeq (\overline{T})\ \{\varphi \,|\, \overline{M}\}\ (\simeq\text{-SIGNATURE})$$

$$\frac{\mathcal{S},\Gamma \vdash p.\mathbf{type} \ni \mathtt{type}_n A}{\mathcal{S},\Gamma \vdash p.A \simeq p.A}\ (\simeq\text{-ABSTYPE})$$

$$\mathcal{S},\Gamma \vdash p.\mathbf{type} \simeq p.\mathbf{type}\ (\simeq\text{-SINGLETON})$$

Algorithmic Subtyping

$$\frac{\mathcal{S},\Gamma \vdash T \simeq T' \qquad \mathcal{S},\Gamma \vdash U \simeq U' \qquad \mathcal{S},\Gamma \vdash_* T' <: U'}{\mathcal{S},\Gamma \vdash T <: U}\ (<:\text{-UNALIAS})$$

$$\frac{A \neq A' \qquad \{n\}\cup\mathcal{S},\Gamma \vdash T_i <: p'.A' \qquad n \notin \mathcal{S} \qquad \mathcal{S},\Gamma \vdash p.\mathbf{type} \ni \mathtt{trait}_n A\, \mathtt{extends}\ (\overline{T})\ \{\varphi \,|\, \overline{M}\}}{\mathcal{S},\Gamma \vdash_* p.A <: p'.A'}\ (<:\text{-CLASS})$$

$$\frac{\mathcal{S},\Gamma \vdash p \simeq p' \qquad \mathcal{S},\Gamma \vdash q \simeq p'}{\mathcal{S},\Gamma \vdash_* p.\mathbf{type} <: q.\mathbf{type}}\ (<:\text{-SINGLETON-RIGHT})$$

$$\frac{\mathcal{S},\Gamma \vdash T_i <: p.A}{\mathcal{S},\Gamma \vdash_* (\overline{T})\ \{\varphi \,|\, \overline{M}\} <: p.A}\ (<:\text{-SIG-LEFT})$$

$$\frac{U \text{ is not a singleton type} \qquad \mathcal{S},\Gamma \vdash p \simeq q \qquad \mathcal{S},\Gamma \vdash_{path} q : T \qquad \mathcal{S},\Gamma \vdash T <: U}{\mathcal{S},\Gamma \vdash_* p.\mathbf{type} <: U}\ (<:\text{-SINGLETON-LEFT})$$

$$\frac{T \text{ is not a singleton type} \qquad \forall i,\ \mathcal{S},\Gamma \vdash T <: T_i \qquad \mathcal{S},\Gamma \vdash T \prec_\varphi \overline{N} \qquad \mathrm{dom}(\overline{M}) \subseteq \mathrm{dom}(\overline{N}) \qquad \mathcal{S},\Gamma \vdash \overline{N} \ll \overline{M}}{\mathcal{S},\Gamma \vdash_* T <: (\overline{T})\ \{\varphi \,|\, \overline{M}\}}\ (<:\text{-SIG-RIGHT})$$

$$\frac{\mathcal{S},\Gamma \vdash p \simeq p' \qquad \mathcal{S},\Gamma \vdash q \simeq p'}{\mathcal{S},\Gamma \vdash_* p.A <: q.A}\ (<:\text{-PATHS})$$

Member Subtyping

$$\frac{\mathcal{S},\Gamma \vdash T <: T'}{\mathcal{S},\Gamma \vdash \mathtt{val}_n a : T\ (= t)^? <: \mathtt{val}_m a : T'\ (= t')^?}\ (<:\text{-MEMBER-FIELD})$$

$$\mathcal{S},\Gamma \vdash \mathtt{type}_n A = T <: \mathtt{type}_n A\ (= T)^?\ (<:\text{-MEMBER-TYPE})$$

$$\mathcal{S},\Gamma \vdash \mathtt{trait}_n A\, \mathtt{extends}\ (\overline{T})\ \{\varphi \,|\, \overline{M}\} <: \mathtt{trait}_n A\, \mathtt{extends}\ (\overline{T})\ \{\varphi \,|\, \overline{M}\}\ (<:\text{-MEMBER-CLASS})$$

$$\frac{\mathcal{S},\Gamma \vdash \overline{S'} <: \overline{S} \qquad \mathcal{S},\Gamma \vdash T <: T'}{\mathcal{S},\Gamma \vdash \mathtt{def}_n a\ (\overline{x : S}) : T\ (= t)^? <: \mathtt{def}_m a\ (\overline{x : S'}) : T'\ (= t')^?}\ (<:\text{-MEMBER-METHOD})$$

Fig. 8. The FS_{alg} Calculus : Subtyping

Both relations use the concept of *evaluation environment* Σ, which is a list of bindings $x : \overline{M}$ that associates an object name x with its set of members. The reduction relation $\Sigma\ ;\ t \rightarrow \Sigma'\ ;\ t'$ reduces t to t' in the environment Σ. The reduction of a term can imply the creation of new objects that are added to the environment, leading to a new environment Σ'. The lookup relation $\Sigma \vdash T \prec_{\varphi} \overline{M}$ collects all declarations $\overline{M}$ in a type T. Together with the concept of evaluation context for terms and types, the rule RED-CONTEXT lets us reduce inside a term. The evaluation of a program $\texttt{val}\ z = \texttt{new}\ \{\varphi \,|\, \overline{M}\}\,; t$ consists in repeatedly reducing the term t in the environment context $z : [z/\varphi]\overline{M}$ until reaching a term that is a value, i.e. a variable y. Such a semantics is needed if we want to state and prove a theorem of type safety. Note that in this semantics the value attached to a field member is re-evaluated each time the field is selected, which corresponds to a call-by-name semantics.

3.3 Type System

The type system of FS_{alg} is described by an algorithmic system of inference rules. In such a system, any judgment is matched by the conclusion of at most one rule, which means that the application of rules is completely deterministic. Typing FS_{alg} requires the definition of several auxiliary judgments about types in addition to the classical judgment that assigns a type T to a term t: membership ($\mathcal{S}, \Gamma \vdash T \ni M$), subtyping ($\mathcal{S}, \Gamma \vdash T <: U$), expansion ($\mathcal{S}, \Gamma \vdash T \prec_{\varphi} \overline{M}$) and type well-formedness ($\mathcal{S}, \Gamma \vdash T$ WF). Most of the judgments are parameterized by a typing context $\mathcal{S}, \Gamma$, where $\mathcal{S}$ is a set of indices representing locked declarations, and Γ is a set of bindings $x : T$ between variables and types such that all variables x are pairwise distinct.

Type assignment. The first two boxes of Figure 6 present the judgments that assign types to terms. The judgment $\mathcal{S}, \Gamma \vdash t : T$ always assigns the most precise type to a term, in particular a path p always receives the type p.**type** according to this judgment, provided it can be assigned a bound T (rule PATH). A field selection $t.a$ is typable if it is possible to type t and to establish, using the *membership* judgment that the type S of t contains a field declaration $\texttt{val}_n a : T\,(= u)^?$; in this case type T is assigned to the selection $t.a$ (rule SELECT). Note that this rule is only applicable if t is *not* a path. In case t is a path p, we fall back to rule PATH. Rule METHOD allows to type method calls $s.a\,(\overline{t})$: the type of s must contain a declaration for the method a, and each argument must have a type which is compatible with the one required by the method declaration, i.e. which is a *subtype* of the expected type. If these conditions are fulfilled, the type announced by the method declaration is given to the term $s.a\,(\overline{t})$. Finally, rule NEW allows the typing of a local object creation $\texttt{val}\ x = \texttt{new}\ T; t$. Such a term gets the type of t if several conditions are satisfied. The members declared inside type T must be concrete: the *expansion* judgment $\mathcal{S}, \Gamma \vdash T \prec_{\varphi} \overline{M}$ returns all the member declarations $\overline{M}$ of a type T such that x represents the self reference. The type T must be *well-formed*. The term t must be typable in the environment Γ extended with the binding $x : T$. Finally, because the scope of x is limited to t, the variable x must not appear in the type S of t.

The judgment $\mathcal{S}, \Gamma \vdash_{path} p : T$ is a typing judgment specialized for paths. Contrary to the previous judgment, it does not always return the singleton type p.**type** for a path p. Rather, it returns the less precise but more informative type associated with it, called its *bound*. If p is a variable x, its bound is the type associated with x in the environment (rule PATH-VAR), if p is a selection $p'.a$, its bound is the declared type T of a as seen from p' (rule PATH-SELECT).

Membership and expansion. In FJ, there is a lookup relation that computes the most precise signature of a method visible from a given class. In FS_{alg} the member judgment $\mathcal{S}, \Gamma \vdash T \ni M$ presented at the bottom of Figure 6 generalizes this relation to the computation of any kind of declaration (not just methods) visible from any kind of type (not just class types). In FJ the lookup relation is quite simple, it returns the signature of the method as it appears in the program; in FGJ, since classes can have type parameters and since a member signature can refer to some type parameters of its enclosing class, the lookup relation requires also the computation of type values for such type parameters. In FS_{alg}, things are more complicated: because types depend on paths, the signature of a member can depend on the self reference x of its enclosing class, or more generally on the self reference of any enclosing class, direct or indirect. Thus, the result of the lookup must replace x with the actual value of the enclosing instance. To illustrate this, suppose we have a method declaration $\mathtt{def}_n a\,() : x.A$ in a class C where x is the self reference. If the starting type T of the membership judgment is a singleton type $p.\mathbf{type}$, then it is possible to replace x with p and obtain $p.A$. But if the starting type T is a class type $q.C$, then the lookup fails because there is no available instance of C with which to replace x. However, if the signature of a does not depend on x (for instance if the return type of a is an external type $root.Int$), then the lookup succeeds, even from $q.C$ (because in $root.Int$ there is no self reference to be replaced). Rules $\ni$-SINGLETON and $\ni$-OTHER respectively implement the situation where the starting type T is a singleton type and where it is not. In the first case, the path p is first expanded into q with the *path alias expansion* judgment. Then, we take the type T of q which, by construction, is not a singleton type. Using the *type expansion judgment* we collect all declarations $\overline{M}$ of T and we substitute p for the self reference φ in the declaration we are interested in. In case the starting type T is not a singleton type, we can immediately collect its declarations, but we have to check that the declaration we are looking for does not contain the self reference φ.

The type expansion judgment $\mathcal{S}, \Gamma \vdash T \prec_\varphi \overline{M}$ (third box of Figure 6) collects all declarations $\overline{M}$ of a type T where φ is used to represent the self reference inside declarations $\overline{M}$. The expansion of a class type $p.A$ is the expansion of the type signature $\left(\overline{T}\right) \left\{\varphi \,|\, \overline{M}\right\}$ composed of its parents $\overline{T}$ and its direct members $\overline{M}$ (rule $\prec$-CLASS). The index n of the class is added to the set $\mathcal{S}$ of locks in order to avoid falling in an infinite expansion (for instance if a class extends itself). Rule $\prec$-TYPE is completely analogous. It expands a type alias while performing the same actions on locks in order to prevent infinite alias expansion. Finally, rule $\prec$-SIGNATURE expands a type signature: it starts by expanding all parents $\overline{T}$ and then merges all collected declarations $\overline{N}$ with the direct members $\overline{M}$ of the type signature. The *concatenation with rewriting of common members* $\uplus$ of several sets of declarations is defined by $\overline{M} \uplus \overline{N} = \overline{M}|_{\mathrm{dom}(\overline{M}) \setminus \mathrm{dom}(\overline{N})}, \overline{N}$, where the domain $dom(\overline{M})$ of a sequence of declarations is the set of labels it defines and the restriction $\overline{M}|_{\mathcal{L}}$ of declarations $\overline{M}$ to a set of labels $\mathcal{L}$ consists of all of those declarations in $\overline{M}$ that define labels in $\mathcal{L}$.

Type and path alias expansion. We introduce here two auxiliary judgments, *type alias expansion* and *path alias expansion* that will be used when defining the subtyping judgment. The idea of the type alias expansion judgment $\mathcal{S}, \Gamma \vdash T \simeq U$ is very simple: we take a type T and if T is a type alias $p.A$ for another type T', we recursively expand T' until we reach a type that is no longer a type alias. This simple behavior is formalized by the five rules in the first box of Figure 8.

There exists also a relation of aliasing between paths. For instance, with a field declaration $\texttt{val}_n a : p.\mathbf{type}$ in a class where x is the self reference, the path $x.a$ has type $p.\mathbf{type}$. Because $p.\mathbf{type}$ is a singleton type and $x.a$ belongs to this type, this really means that $x.a$ and p represent the same object, or equivalently that $x.a$ is an *alias* for p. In this reasoning, we have performed a one-step alias expansion going from $x.a$ to p. The judgment defined at the bottom of Figure 7 implements the complete alias expansion of a path by repeating the operation we have performed in this example. Once again, we prevent falling into a loop by adding an index to the set $\mathcal{S}$ of locks (rule $\simeq$-STEP). This index is the one of the last field selected in the considered path p. It is computed by the function $\psi(p)$. This function takes as implicit arguments the environments $\mathcal{S}$ and Γ and is defined as follows.

$$\begin{aligned} \psi(p.a) &= n \quad \text{if } \mathcal{S}, \Gamma \vdash p.\mathbf{type} \ni \texttt{val}_n a : T\,(= t)^? \\ \psi(x) &= x \end{aligned}$$

The expansion terminates when we reach a path that cannot be given a singleton type (rule $\simeq$-REFL).

Subtyping. The subtyping judgment $\mathcal{S}, \Gamma \vdash T <: U$ (central box of Figure 8) is used to compare two types T and U. In theory, such a relation must be defined by considering all possible kinds of types for T and all possible kinds of types for U. But in practice it is possible to factorize and eliminate a great number of cases. First we expand both types into types T' and U' (rule <:-UNALIAS). This simple operation allows to quickly eliminate all cases where T' or U' is an abstract type $p.A$, because if a type is still abstract after alias expansion nothing can be said about it. As a consequence, in the auxiliary judgment $\mathcal{S}, \Gamma \vdash_* T <: U$ we assume that T and U are not abstract types. If U is a singleton type $q.\mathbf{type}$, then only another singleton type $p.\mathbf{type}$ can be a subtype of it (rule <:-SINGLETON-RIGHT), and in this case paths p and q must be equivalent, i.e. they must be aliases for the same path p'. If the left-hand side is a singleton type $p.\mathbf{type}$ and the right-hand side U is not, then we just take the bound T of p and recursively compare it with U (rule <:-SINGLETON-LEFT). If both types are a selection of the same type label A (rule <:-SINGLETON-LEFT), then their prefixes must be equivalent. Suppose now that the right-hand side is a class type. We just have to consider the cases where the left-hand side is a class type or a type signature. If the left hand-side is a class type $p.A$ then we recursively check that there exists one parent of the class that is a subtype of $p'.A'$ (rule <:-CLASS). If the left-hand side is a type signature $\left(\overline{T}\right)\left\{\varphi \mid \overline{M}\right\}$ the procedure is analogous. Finally, we are left with the case where the right-hand side is a type signature $\left(\overline{T}\right)\left\{\varphi \mid \overline{M}\right\}$. There are then two things to check: first that the type T is a subtype of all parents $\overline{T}$ in the type signature, which expresses the fact that a type signature represents the intersection of its parents. And secondly, that type T satisfies the constraints expressed by the declarations $\overline{M}$. This is the case if T expands to a set of declarations $\overline{N}$, with a greater domain than $\overline{M}$, and if the declarations that are common to $\overline{N}$ and $\overline{M}$ are more precise in $\overline{N}$, which is expressed by the subtyping test between members $\mathcal{S}, \Gamma \vdash \overline{N} \ll \overline{M}$.

The subtyping between members is standard (bottom of Figure 8): it is covariant for the types of field declarations and for the result types of methods, contravariant for the method parameter types, and invariant for type and class declarations. The invariance for type aliases is crucial since an alias conceptually represents an equality between types. The member subtyping relation is lifted to sequences of members using the following definitions:

$$\overline{N} \ll \overline{N'} \Leftrightarrow \left(\forall (N, N') \in \overline{N} \times \overline{N'},\ \mathrm{dom}(N) = \mathrm{dom}(N') \Rightarrow N <: N'\right)$$

Well-formedness. There are two well-formedness judgments: one for types $\mathcal{S}, \Gamma \vdash T\ \textsc{wf}$ (top of Figure 7), and one for members $\mathcal{S}, \Gamma \vdash M\ \textsc{wf}_\varphi$ (top of Figure 7)

For a singleton type $p.\mathbf{type}$ to be well-formed, the path p must be typable (rule WF-SINGLETON). In order to avoid a cyclic dependence between a path and its type, we also check the well-formedness of the bound T of p. Because we do not want to fall into a loop, we extend the set of locked indices with $\psi(p)$. For a class type $p.A$ to be well-formed (rule WF-CLASS), it is sufficient to check that a class named A is accessible from p, which is expressed by a membership judgment starting from the type $p.\mathbf{type}$. For an abstract type $p.A$, a declaration of the type A must also be visible from p. In addition, if the type is an alias for a type T we also check that T is well-formed. This is needed since we do not want type aliases to let us define recursive types. Such a test serves for detecting illegal cycles, as for the rule WF-SINGLETON. Finally, the well-formedness of a type signature is checked by first putting a binding with the type signature in the environment (rule WF-TYPE).

The well-formedness of fields and methods is standard. For a field declaration the type T' of the optional value t must conform to the declared bound T (rule WF-X-FIELD). For a method, the types of parameters $\overline{S}$ and the method type T must be well-formed in the current environment, which means that the judgment excludes the possibility for the type of a parameter to depend on another parameter. The body t of the method must be typable in an environment extended with the parameters, and its type T' must conform to the return type (rule WF-X-METHOD). Note that we require parameter types not to contain singleton types. This restriction has almost no impact on expressiveness and it has the advantage of simplifying the termination proof of path alias expansion. The well-formedness of a type declaration is equivalent to the well-formedness of the type T it is an alias for (rule WF-X-TYPE). Eventual cycles in this definition are detected indirectly by the well-formedness of T. For typing a class declaration (rule WF-X-CLASS) we check the well-formedness of its associated type signature $(\overline{T})\ \{\varphi \mid \overline{M}\}$ composed of its parents and direct members, in an environment extended with a binding for the class self reference φ. It might be surprising that here we do not check for the absence of cycles in the class hierarchy, but such cycles are actually detected by the well-formedness of the type signature: if a class inherits, directly or indirectly, of itself, there cannot exist an expansion of the parents $\overline{T}$. A type signature $(\overline{T})\ \{\varphi \mid \overline{M}\}$ (rule WF-X-SIGNATURE) is well-formed w.r.t. a self reference φ if the $\overline{T}$ can be expanded, which forbids singleton types and abstract types in the parents of a signature, and if all new members of the type, directly present in $\overline{M}$, are compatible. The compatibility of a list of groups of members is defined in such a way that a member declaration is always more precise than another declaration defined in a previous group in this list, which is expressed by the formula $\forall\ (i,j),\ \mathcal{S}, \Gamma \vdash (\overline{N_{i+j}}, \overline{M}) \ll \overline{N_i}$.

4 Decidability of the Algorithmic Type System

Lemma 4.1. If a term t can be assigned a type T by the **Path Typing** judgment, then it is unique.

Proof. It is easy to see that a variable only has a single type assignement in the context at any time, so all that remains to prove is that field declarations of the form $\mathtt{val}_n a : T\,(= t)^?$ for a given a are unique in a given $p.$**type**. We do it by induction on the **Expansion** judgment and PATHSELECT, using the semantics of $\uplus$, a method that we are going to detail in the following.

Lemma 4.2. The calculus defines a deterministic algorithm.

Proof. The rules are syntax-directed (the form of the input determines the rule that must be used, and all the parameters of any recursive calls), except for $\prec$-CLASS, $\prec$-TYPE and $\simeq$-STEP.

$\prec$-CLASS and $\prec$-TYPE seem to create an ambiguity, but they are in fact algorithmically equivalent, since they only differ on the member that will be sought in the expansion of the type of the path that is in the conclusion.

The path in the premises of $\simeq$-STEP is uniquely defined by the **Path Typing** judgment using Lemma 4.1.

Lemma 4.3. The **Path Typing**, **Expansion**, **Membership**, and **Path Alias Expansion** judgments terminate on all inputs.

Proof. We start by inlining the membership rule $\ni$-SINGLETON in the recursive calls of the three other judgments, leading to the following rules:

$$\frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash p \simeq q \qquad \mathcal{S}, \Gamma \vdash_{path} q : T \\ \psi(p) \cup \mathcal{S}, \Gamma \vdash T \prec_{\varphi} \overline{M}, [\varphi/p](\mathtt{val}_n a : T'\,(= t)^?), \overline{M'} \qquad \psi(p) \nsubseteq \mathcal{S}\end{array}}{\mathcal{S}, \Gamma \vdash_{path} p.a : T'}\ (\text{PATH-SELECT})$$

$$\frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash p \simeq q \qquad \mathcal{S}, \Gamma \vdash_{path} q : T \\ \psi(p) \cup \mathcal{S}, \Gamma \vdash T \prec_\varphi \overline{M}, [\varphi/p](\mathtt{trait}_n A\, \mathtt{extends}\, (\overline{S})\; \{x \mid \overline{N}\}, \overline{M'}) \qquad \psi(p) \not\subseteq \mathcal{S} \\ \{n\} \cup \mathcal{S}, \Gamma \vdash (\overline{S})\; \{x \mid \overline{N}\} \prec_x \overline{N'} \qquad n \notin \mathcal{S}\end{array}}{\mathcal{S}, \Gamma \vdash p.A \prec_x \overline{N'}}$$

(≺-CLASS)

$$\frac{\begin{array}{c}\mathcal{S}, \Gamma \vdash p \simeq q \qquad \mathcal{S}, \Gamma \vdash_{path} q : T \\ \psi(p) \cup \mathcal{S}, \Gamma \vdash T \prec_\varphi \overline{M}, [\varphi/p](\mathtt{type}_n A = S), \overline{M'} \qquad \psi(p) \not\subseteq \mathcal{S} \\ \{n\} \cup \mathcal{S}, \Gamma \vdash S \prec_x \overline{N'} \qquad n \notin \mathcal{S}\end{array}}{\mathcal{S}, \Gamma \vdash p.A \prec_x \overline{N'}} \quad (\prec\text{-TYPE})$$

The other rules are left unchanged, and the system with those updated rules is trivially equivalent to the previous one. We prove the termination of the **Expansion**, **Path Typing** and **Path Alias Expansion** judgments by mutual induction, reasoning by case on the last rule of the derivation. The result is then easily extended to the **Membership** judgement by inspection.

Cases ≃-REFL and PATHVAR are easy. For the others, we consider that several rules in those jugements make all their recursive calls on a strictly larger set of locked symbols, and can therefore appear at most a finite number of times in any derivation. We can then concentrate on the remaining case, ≺-SIGNATURE. Let us define the *size* of a type by the lexicographical pair (N, L) where N is the number of its members and L its textual size. Then this size is finite and positive, and ≺-SIGNATURE only makes recursive calls on strictly smaller types. Those recursive calls conclude when reduced to a type signature with no inherited types, in which case the conclusion is true by vacuity.

Corollary 1. *The **Type Alias Expansion** judgment terminates on all inputs.*

Proof. Easy induction, considering that the rule ≃-TYPE can only be applied a finite number of times.

Lemma 4.4. The **Algorithmic Subtyping** and **Member Subtyping** judgments terminate on all inputs.

Proof. We proceed by mutual induction on those two jugements, and then by case on the last step of the derivation. Using Corollary 1, we can concentrate on the subtyping of unaliased terms ($\vdash_*$) using rule <:-UNALIAS.

*Case 1 (**Algorithmic Subtyping**).* Cases <:-SINGLETON-RIGHT and <:-PATHS are easy using Lemma 4.3. Moreover, it is easy to show using the **Path Alias Expansion** judgment, that the type T in the premises of <:-SINGLETON-LEFT is not a singleton type. This rule therefore allows us to *unfold* a subtyping problem referring to a singleton type to a subtyping problem between non-singleton types in a single step. Since our subtyping rules do not contain singleton types in their premises, the only singleton types we can encounter in a subtyping derivation are those explicitly mentioned in the program, and their number is finite. We can therefore consider that we work modulo this *unfolding*.

Using the definition of the size of a type given in the proof of Lemma 4.3, we can show that all the recursive calls to the **Algorithmic Subtyping** judgment are made on a strictly smaller type, noticing in the case of <:-CLASS that a given `trait` can not extend itself.

The remaining case is <:-SIG-RIGHT, where the call to the **Member Subtyping** judgment could potentially create a cycle. Let us proceed by contradiction: if there is such a cycle, it means that we have found a type signature $\left(\overline{T}\right) \left\{x \mid \overline{M}\right\}$ such that one of its member declarations contains a declared type that is is either $\left(\overline{T}\right) \left\{x \mid \overline{M}\right\}$ or some $\left(\overline{S}\right) \left\{y \mid \overline{N}\right\}$ such that $\exists\, i,\ S_i = \left(\overline{T}\right) \left\{x \mid \overline{M}\right\}$. Then the textual length of this type is larger than the textual length of the type signature that contains it, which is absurd.

*Case 2 (**Member Subtyping**).* Easy with the previous case.

Lemma 4.5. The **Type Assignement**, **Well-Formedness** and **Member Well-Formedness** judgments terminate on all inputs.

Proof. We proceed by mutual induction on those jugements, then by case on the last rule of the derivation.

*Case 3 (**Member Well-Formedness**).* Considering WF-X-SIGNATURE, we notice that, in a similar way to the problem we encountered with rule <:-SIG-RIGHT in Case 1 of Lemma 4.4, we are in presence of a judgement that makes several potentially cyclic recursive calls to the **Well-Formedness** judgment. However, since the judgment is syntax-directed, we notice that such a cycle would require us to find a type signature $\left(\overline{T}\right) \left\{x \mid \overline{M}\right\}$ directly containing a member whose declared type lexically contains $\left(\overline{T}\right) \left\{x \mid \overline{M}\right\}$ itself, which would give this type signature an infinite textual length.

The remaining interesting cases are WF-X-FIELD and WF-X-METHOD. in both cases, making a derivation involving those rules cyclic requires defining a type signature that directly contains a field or method whose value contains an instantiation of this very type signature:

$$\overline{M} \ni \texttt{val}_n a : T = \texttt{val}\ \ x = \texttt{new}\ \ \left(\overline{T}\right) \left\{x \mid \overline{M}\right\} ; t$$

This would again give an infinite textual length to the term $\left(\overline{T}\right) \left\{x \mid \overline{M}\right\}$.

*Case 4 (**Well-Formedness**).* We start by noticing that WF-SINGLETON and WF-TYPE can only occur a finite number of times, since they make their recursive calls on a strictly larger set of recursive types. We then conclude using the previous case and the termination lemma for the **Membership** judgment (Lemma 4.3).

*Case 5 (**Type Assignement**).* We conclude remarking that we make recursive calls to the **Type Assignement** judgment on structurally smaller terms, and using the previous termination lemmas.

5 Type Checking and Type Inference in Scala

The previous section showed that type-checking in FS_{alg} is decidable. Does the same hold for full Scala? It is at present too hard to give a definite answer since full Scala is too complicated to admit a formalization of its type system which is complete yet still manageable enough to admit a proof of decidability. But one can conjecture. To do this, we need to compare full Scala with Featherweight Scala. Most of the additional syntactic constructs in full Scala do not cause particular problems for type-checking. However, unlike Featherweight Scala, full Scala has local type inference [30,26].

Local type inference needs to construct least upper bounds and greatest lower bounds (*wrt* the subtype ordering) of sets of types. The decidability of these *lub* and *glb* operations in FS_{alg} is currently an open question. To see the problem, consider the following three class definitions.

```
trait A { this0 |
    type T
    def fromT(x: T): A
}
trait B { this0 |
    type T
    def fromT(x: T): B
}
trait C extends A with B { this0 |
    def fromT(x: T): C
}
```

Now assume that we want to find the greatest lower bound of A and B. Clearly, C is a lower bound of A and B, but it is not the greatest one. A greater lower bound is represented by the following refinement type:

```
A with B { this0 | fromT(x: T): C }
```

One can apply the same step to the result type of *fromT* to obtain a still greater lower bound. Repeating this step infinitely often one obtains the following limit of an ascending chain of lower bounds:

```
A with B { this0 |
    fromT(x: T): A with B { this1 |
        fromT(x: T): A with B { this2 |
            fromT(x: T): A with B { this3 |
                ...
            }
        }
    }
}
```

This limit does not exit as a finite type in FS_{alg}, but the natural algorithm for computing lower bounds is likely to try to construct it, and this would result in non-termination. A similar infinite approximation can be constructed for the *lub* operation by using the contravariance of method parameters. An example is the *lub* of the two refinements

$$\{\ \textbf{\textit{def}}\ f(x\colon A)\colon Boolean\ \} \quad and \quad \{\ \textbf{\textit{def}}\ f(x\colon B)\colon Boolean\ \}\ .$$

The problem of infinite approximations of *lub*'s and *glb*'s also occurs when type-checking Java 1.5 programs with generics and wildcards [34]. The decidability of the latter is currently open [35].

The *scalac* compiler addresses this problem by imposing a maximum size on the types computed by its *lub* and *glb* operations. It is currently set at 10 levels of parameterizations or refinements. If a type computed by *lub* or *glb* exceeds this limit the system will reply with an error such as the one below:

```
error: failure to compute least upper bound of types
(A) ⇒ scala.Int and (B) ⇒ scala.Int;
an approximation is:
(A with B{
    def fromT(this.T): (A with B{
        def fromT(this.T): (A with B{
            def fromT(this.T): (A with B{...})})})}) ⇒ Int
additional type annotations are needed
    if (cond) (x: A) ⇒ 1 else (x: B) ⇒ 1
    ^
```

The Scala compiler thus turns the potential problem of undecidability of type inference into a completeness problem: local type inference might now fail to give a solution even if a best type would exist. However, in practice such complicated types arise very rarely. Moreover, it is always possible to guide the type inference process by adding more type annotations, so that infinite approximations are avoided.

In the failed example above, the problem would have been avoided by giving an explicit annotation of the desired type of the problematic conditional. For instance, the following compiles without error.

```
(if (cond) (x: A) ⇒ 1 else (x: B) ⇒ 1): (C ⇒ int)
```

To summarize, the results on type-checking Featherweight Scala give some degree of confidence that type-checking regular Scala is also decidable. Furthermore, the formalization of locks in FS_{alg} corresponds closely to the present implementation in the Scala compiler, so that there is hope that this implementation does in fact represent an algorithm for type checking Scala programs. Type-inference, on the other hand, needs to compute *lub*'s and *glb*'s of types and is believed to be undecidable. The Scala compiler avoids potential non-termination at the price of incompleteness by imposing an upper limit on the size of the types computed by a *lub* or *glb*.

Note that we have classified here the typing of an if-then-else expression as a type-inference problem, not a type checking problem. The justification of this classification is that it is possible (and, at rare occasions, necessary) to provide a type for the expression with an explicit annotation.

6 Conclusion

We have presented a calculus for type-checking core Scala programs. Featherweight Scala decribes the central constructs for programming components in Scala: nested classes, modular mixin composition, abstract types, type aliases, and path-dependent types. Unlike previous work on foundations of Scala [25], this calculus is decidable and admits a straight-forward type-checking algorithm.

Featherweight Scala programs are essentially a syntactic subset of regular Scala programs. The subset is kept minimal, so that one can concentrate on a small set of typing issues. In future work it would be interesting to extend the calculus to a larger fragment of Scala. Among the most interesting extensions are a call-by-value semantics, polymorphic methods, and mutable state.

The correctness of the calculus also remains to be verified. The operational semantics of FS_{alg} is defined by a small-step reduction semantics. We intend to show in future work that it satisfies the subject-reduction and type-soundness properties. Judging from our experience with previous calculi [25,10] this looks plausible, but a formal proof still needs to be completed.

Acknowledgement. We thank Rachele Fuzzati for proofreading the formal description of the calculus.

References

1. P. Altherr. *A Typed Intermediate Language and Algorithms for Compiling Scala by Successive Rewritings.* PhD thesis, EPFL, March 2006. No. 3509.
2. P. Altherr and V. Cremet. Inner Classes and Virtual Types. EPFL Technical Report IC/2005/013, March 2005.
3. D. Ancona and E. Zucca. A primitive calculus for module systems. In *Principles and Practice of Declarative Programming*, LNCS 1702, 1999.
4. D. Ancona and E. Zucca. A calculus of module systems. *Journal of Functional Programming*, 2002.
5. G. Bracha. *The Programming Language Jigsaw: Mixins, Modularity and Multiple Inheritance.* PhD thesis, University of Utah, 1992.
6. G. Bracha and G. Lindstrom. Modularity meets inheritance. In *Proceedings of the IEEE Computer Society International Conference on Computer Languages*, pages 282–290, Washington, DC, 1992. IEEE Computer Society.
7. K. Bruce. Some challenging typing issues in object-oriented languages. In *Electronic notes in Theoretical Computer Science, volume 82(8).*, 2003.
8. K. B. Bruce, M. Odersky, and P. Wadler. A statical safe alternative to virtual types. In *Proceedings of the 5th International Workshop on Foundations of Object-Oriented Languages*, San Diego, USA, 1998.
9. K. Crary, R. Harper, and S. Puri. What is a recursive module? In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 50–63, 1999.
10. V. Cremet. *Foundations for Scala: Semantics and Proof of Virtual Types.* PhD thesis, EPFL, May 2006. No. 3556.
11. E. Ernst. *gBeta: A language with virtual attributes, block structure and propagating, dynamic inheritance.* PhD thesis, Department of Computer Science, University of Aarhus, Denmark, 1999.

12. E. Ernst. Family polymorphism. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 303–326, Budapest, Hungary, 2001.
13. E. Ernst, K. Ostermann, and W. Cook. A virtual class calculus. In *ACM Symposium on Principles of Programming Languages (POPL'06)*, Jan. 2006.
14. J. Garrigue. Code reuse through polymorphic variants. In *In Workshop on Foundations of Software Engineering, Sasaguri, Japan, November 2000.*, 2000.
15. R. Harper and M. Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages*, January 1994.
16. T. Hirschowitz and X. Leroy. Mixin modules in a call-by-value setting. In *European Symposium on Programming*, pages 6–20, 2002.
17. A. Igarashi and B. C. Pierce. Foundations for virtual types. *Information and Computation*, 175(1):34–49, 2002.
18. A. Igarashi and B. C. Pierce. On inner classes. *Inf. Comput.*, 177(1):56–89, 2002.
19. A. Igarishi, B. Pierce, and P. Wadler. Featherweight Java: A minimal core calculus for Java and GJ. In *Proc. OOPSLA*, Nov. 1999.
20. P. Jolly, S. Drossopoulou, C. Anderson, and K. Ostermann. Simple dependent types: Concord. In *Proc. FTfJP*, 2004.
21. X. Leroy. A syntactic theory of type generativity and sharing. In *ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon*, 1994.
22. O. L. Madsen, B. Møller-Pedersen, and K. Nygaard. *Object-Oriented Programming in the BETA Programming Language.* Addison-Wesley, June 1993. ISBN 0-201-62430-3.
23. K. Nakata, A. Ito, and J. Garrigue. Recursive object-oriented modules. In *Proc. FOOL 12*, Jan. 2005.
24. N. Nystrom, S. Chong, and A. Myers. Scalable extensibility via nested inheritance. In *Proc. OOPSLA*, pages 99–115, 2005.
25. M. Odersky, V. Cremet, C. Röckl, and M. Zenger. A nominal theory of objects with dependent types. In *Proc. ECOOP'03*, Springer LNCS, July 2003.
26. M. Odersky, C. Zenger, and M. Zenger. Colored local type inference. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, pages 41–53, January 2001.
27. M. Odersky and M. Zenger. Independently extensible solutions to the expression problem. In *Proc. FOOL 12*, Jan. 2005. `http://homepages.inf.ed.ac.uk/wadler/fool`.
28. M. Odersky and M. Zenger. Scalable component abstractions. In *Proc. OOPSLA*, 2005.
29. B. C. Pierce. Bounded quantification is undecidable. *Information and Computation*, 112(1):131–165, July 1994.
30. B. C. Pierce and D. N. Turner. Local type inference. In *Proc. POPL*, 1998.
31. D. Rémy and J. Vuillon. On the (un)reality of virtual types. available from `http://pauillac.inria.fr/remy/work/virtual`, Mar. 2000.
32. M. Torgersen. Virtual types are statically safe. In *5th Workshop on Foundations of Object-Oriented Languages*, San Diego, CA, USA, January 1998.
33. M. Torgersen. The expression problem revisited: Four new solutions using generics. In *Proc. ECOOP 2004*, volume 3086 of *Springer LNCS*, pages 123–143. Springer-Verlag, July 2004.
34. M. Torgersen, C. P. Hansen, E. Ernst, P. vod der Ahé, G. Bracha, and N. Gafter. Adding wildcards to the Java programming language. In *Proceedings SAC'04*, pages 1289–1296, Nicosia, Cyprus, Mar. 2004. ACM Press.
35. S. Zdancewic. Type inference for Java 5: Wildcards, F-bounds, and undecidability. `http://www.cis.upenn.edu/~stevez/note.html`, 2006.

Tree Exploration with an Oracle

Pierre Fraigniaud[1], David Ilcinkas[2], and Andrzej Pelc[3]

[1] CNRS, Laboratoire de Recherche en Informatique (LRI)
Université Paris-Sud
91405 Orsay, France
`pierre@lri.fr`

[2] Laboratoire de Recherche en Informatique (LRI)
Université Paris-Sud
91405 Orsay, France
`ilcinkas@lri.fr`

[3] Département d'informatique, Université du Québec en Outaouais
Gatineau, Québec J8X 3X7, Canada
`pelc@uqo.ca`

Abstract. We study the amount of knowledge about the network that is required in order to efficiently solve a task concerning this network. The impact of available information on the efficiency of solving network problems, such as communication or exploration, has been investigated before but assumptions concerned availability of *particular* items of information about the network, such as the size, the diameter, or a map of the network. In contrast, our approach is *quantitative*: we investigate the minimum number of bits of information (minimum oracle size) that has to be given to an algorithm in order to perform a task with given efficiency.

We illustrate this quantitative approach to available knowledge by the task of tree exploration. A mobile entity (robot) has to traverse all edges of an unknown tree, using as few edge traversals as possible. The quality of an exploration algorithm $\mathcal{A}$ is measured by its *competitive ratio*, i.e., by comparing its cost (number of edge traversals) to the length of the shortest path containing all edges of the tree. Depth-First-Search has competitive ratio 2 and, in the absence of any information about the tree, no algorithm can beat this value.

We determine the minimum number of bits of information that has to be given to an exploration algorithm in order to achieve competitive ratio strictly smaller than 2. Our main result establishes an exact threshold oracle size that turns out to be roughly $\log\log D$, where D is the diameter of the tree. More precisely, for any constant c, we construct an exploration algorithm with competitive ratio smaller than 2, using an oracle of size at most $\log\log D - c$, and we show that every algorithm using an oracle of size $\log\log D - g(D)$, for any function g unbounded from above, has competitive ratio at least 2.

1 Introduction

For many network problems (such as leader election, minimum spanning tree, rendezvous, wakeup, broadcasting, etc.), the quality of the algorithmic solutions often depends on the amount of knowledge given to nodes of the network, or given to mobile entities moving in the network, about its topology. *Local* knowledge given to every node and/or to every mobile entity is its identity and, for a node, its degree (or the list of neighbor identities). Any other knowledge (e.g., the total number of nodes, network diameter, the total number of mobile entities, partial maps of the network, etc.) is *global* knowledge. Many results illustrate the impact of global knowledge on the ability and efficiency of solving network problems. For instance, it is proved in [4] that, if an upper bound $\hat{n}$ on the number n of nodes of a graph is known, then a robot can explore this graph in time polynomial in $\hat{n}$, using one pebble, while without this knowledge, $\Theta(\log\log n)$ pebbles are necessary and sufficient. Broadcasting in radio networks is another subject where global information significantly influences efficiency. In [22] it is shown that if nodes have complete knowledge of the network then deterministic broadcasting can be done in time $O(D + \log^3 n)$, for n-node radio networks with diameter D. (This result has been recently improved to $O(D + \log^2 n)$ in [24]). On the other hand, in [9] a lower bound of $\Omega(n \log D)$ is proved on deterministic broadcasting time in radio networks in which nodes know only their own identity. (An almost matching upper bound of $O(n \log^2 D)$ is proved in [10]). In fact, the impact of global knowledge is significant in many areas of distributed computing, as witnessed by [19,25] where hundreds of impossibility results and lower bounds for distributed computing are surveyed, many of them depending on whether or not the nodes are given exact or approximate values of global parameters providing partial knowledge of the topology of the network. Finally, notice that the amount of global knowledge has also a strong impact on computing in anonymous networks (cf., e.g., [23], where the impact of knowing the total number of nodes is studied in depth).

We model global knowledge, given to the nodes or to the mobile entities, by an *oracle*. Given a problem $\mathcal{P}$ with the set of instances $\mathcal{I}$, an oracle is a function $\mathcal{O} : \mathcal{I} \mapsto \{0,1\}^*$ that maps any instance I to a binary string $\mathcal{O}(I)$. Solving problem $\mathcal{P}$ using oracle $\mathcal{O}$ consists in designing an algorithm that, given the binary string $\mathcal{O}(I)$, but unaware of I, returns a *$\mathcal{P}$-scheme* for I, i.e., a sequence of instructions executed by the nodes or the mobiles entities, solving $\mathcal{P}$ for I. In this setting, the amount of global knowledge is measured by the *size* of the oracle on every instance I, i.e., the length of the binary string $\mathcal{O}(I)$. Typical questions of interest are then: "What is the minimum size of an oracle for solving problem $\mathcal{P}$?" or "What is the minimum size of an oracle for solving $\mathcal{P}$ within some amount of time?". The novelty and significance of our modeling of global knowledge is that it enables asking such *quantitative* questions about the required knowledge, regardless of what *kind* of knowledge is supplied. This should be contrasted with the traditional approach that assumes availability of particular items of global information.

Modeling knowledge about the network by an oracle has already proved useful in the context of communication problems. In a recent paper [21], we showed tight bounds on oracle size required for an efficient execution of two fundamental communication tasks: broadcast and wakeup. It turns out that the minimum oracle size required for broadcast with a linear number of messages is strictly larger than that required for wakeup with a linear number of messages. In this paper, we address similar quantitative questions about knowledge required for one of the fundamental problems in mobile computing: the exploration problem. We prove a tight bound of roughly $\log \log D$ on the size of an oracle enabling the design of an exploration algorithm with competitive ratio strictly less than 2, on trees of diameter D.

1.1 The Background of Tree Exploration

A robot has to traverse all edges of an undirected connected graph, using as few edge traversals as possible. Graph exploration is most often performed when the robot lacks some essential information on the explored graph. In such case, the quality of an exploration algorithm $\mathcal{A}$ is measured by comparing its cost (number of edge traversals) to the length of the shortest *covering walk* (i.e., the shortest path containing all edges of the graph). This ratio, maximized over all graphs and all starting nodes, is called the *competitive ratio* $\mathcal{R}(\mathcal{A})$ of algorithm $\mathcal{A}$. The situation here is similar to the context of online algorithms, where competitive ratio first appeared. In both cases, the performance of an algorithm lacking some essential knowledge about the environment is compared to that of an algorithm that has this knowledge: in the case of online algorithms, this knowledge concerns future events, and in the case of exploration, it concerns the topology of the graph and its labeling. (An algorithm provided with a fully labeled copy of the explored graph, showing which port at a visited node leads to which neighbor, can find the shortest covering walk off line.)

Depth-First-Search has competitive ratio 2 and it was shown in [14] that no exploration algorithm can beat this value for arbitrary graphs, even when provided with an unlabeled isomorphic copy of the explored graph with the starting node marked. It turns out that merely the absence of labels of ports and nodes in the map is sufficient to confuse any algorithm on some graphs, making it not better than DFS. On the other hand, in the absence of any global information whatsoever, beating competitive ratio 2 was shown impossible even for the family of trees. This leads to the question if competitive ratio smaller than 2 is possible to achieve for tree exploration, if the algorithm is provided with some partial information concerning the explored environment. In [14] a positive answer to this question was given in the case of very large additional information: the robot was provided with an unlabeled map of the tree. However, this assumption is not very realistic. Indeed, exploration is often used as a tool to construct a map of an unknown network, and usually a priori information about the explored network is much more restricted.

1.2 The Problem

We consider the problem of the *amount of information* needed to achieve tree exploration with competitive ratio smaller than 2. (Recall that the reason of restricting attention to trees is the above mentioned negative result for general graphs, showing that already relatively simple graphs force competitive ratio at least 2 even with extensive additional information, namely an entire unlabeled copy of the explored graph.)

The problem is formalized as follows. In the framework of tree exploration, we define an *oracle* to be a function $\mathcal{O}$ from the class of all trees to the class of binary strings. Specifically, for every tree T, an exploration algorithm is provided with the string $\mathcal{O}(T)$ and returns an *exploration scheme* for T. Such a scheme, starting at any node u, traverses all edges of T. The size of the oracle for tree T is the length of the string $\mathcal{O}(T)$. We ask what is the minimum size of an oracle for which there exists an exploration algorithm achieving competitive ratio smaller than 2, for all trees.

1.3 Our Results

We use the notion of oracle to measure the minimum amount of information required for the design of an efficient exploration algorithm. Our main result establishes an exact threshold oracle size to achieve competitive ratio smaller than 2 for tree exploration. This threshold turns out to be roughly $\log\log D$, where D is the diameter of the tree. More precisely, for any constant c we construct an exploration algorithm with competitive ratio smaller than 2, using an oracle of size at most $\log\log D - c$, and we show that every algorithm using an oracle of size $\log\log D - g(D)$, for any function g unbounded from above, has competitive ratio at least 2.

It is interesting to note the *structure* of the oracle in our positive result. For any tree T, this is a string s of bits depending only on D, and giving an approximation of it, plus an additional bit b that allows the robot to choose between two types of exploration. This additional bit b (depending on D and on the size of the tree) is very important. Indeed, while the string s depends only on D and has length smaller than $\log\log D$, we show that even the full knowledge of D, but without b, is not sufficient to beat competitive ratio 2. More precisely, we show that every exploration algorithm knowing only the diameter of the tree must have competitive ratio at least 2.

1.4 Related Work

Exploration of unknown environments has been extensively studied in the literature, both in the geometric and in the graph setting. In the first scenario the environment is modeled, e.g., as a terrain with obstacles that may be convex [7], polygonal [11] or rectangular [3]. Another way is to represent the unknown environment as a graph, assuming that the robot may only move along its edges. The graph model is further specified in two different ways. In [1,4,5,13,20] the robot explores strongly connected directed graphs and it can move only in the direction from tail to head of an edge, not vice-versa. In [1,13] the authors study competitive ratio of algorithms exploring directed graphs. The constructed algorithms have competitive ratio exponential in the deficiency d of the graph [13], or competitive ratio $d^{O(\log d)}m$, where m is the number of edges [1]. Recently, the first exploration algorithm with competitive ratio polynomial in the deficiency of the graph has been given in [20].

In [2,8,14,18,26,27] the explored graph is undirected and the robot can traverse edges in both directions. In some papers additional restrictions on the moves of the robot are imposed. It is assumed that the robot has either a restricted tank [2,8], forcing it to periodically return to the base for refueling, or that it is tethered, i.e., attached to the base by a rope or cable of restricted length [18].

Another direction of research concerns exploration of anonymous graphs (directed or undirected). In this case it is impossible to explore arbitrary graphs and stop, if no marking of nodes is allowed. Hence the scenario adopted in [4,5] is to allow pebbles which the robot can drop on nodes to recognize already visited ones, and then remove them and drop in other places. The authors concentrate attention on the minimum number of pebbles allowing efficient exploration of arbitrary directed graphs. Exploring anonymous trees without the possibility of marking nodes is investigated in [15]. The authors concentrate attention not on the cost of exploration but on the minimum amount of memory sufficient to carry out this task. Exploration of anonymous graphs was also considered in [12,16,17].

2 Terminology and Preliminaries

For any tree T we denote by $|T|$ the number of nodes of T, and call it the *size* of this tree. For a given tree T and starting node u, we denote by $opt(T, u)$ the length of the shortest covering walk of T starting from u, i.e., the length of the shortest path in T starting from u and containing all edges of T. Clearly, $opt(T, u) = 2(n-1) - ecc(u)$, where n is the size of T and $ecc(u)$ is the eccentricity of the starting node u, i.e., the distance from u to the farthest leaf. Depth-First-Search ending in the leaf farthest from the starting node u uses fewest edge traversals.

We assume that all ports at a node v are numbered 1,...,deg(v). Hence the robot can recognize already visited nodes and traversed edges. However, it cannot tell the difference between yet unexplored edges incident to its current position. The robot executes a given *exploration scheme* that, at every node v, makes one of the following decisions: take a specific already explored edge, or take an unexplored edge. If the scheme decides to take an unexplored edge, the actual choice of the edge belongs to an adversary, as we are interested in worst-case performance.

We want an oracle to provide information on the topology of the explored tree, independently of any labeling, hence we define it as a function $\mathcal{O}$ from the class of all *unlabeled* trees to the class of binary strings. For any string s, a tree T such that $\mathcal{O}(T) = s$ is called *compatible* with s. If a tree exploration algorithm $\mathcal{A}$ takes the string $\mathcal{O}(T)$ as input for any tree T, we say that $\mathcal{A}$ *uses* $\mathcal{O}$.

Consider an exploration algorithm $\mathcal{A}$ using oracle $\mathcal{O}$. For any string s in the range of $\mathcal{O}$, algorithm $\mathcal{A}$ produces an exploration scheme that explores all trees compatible with s. For any such tree T and starting node u, the *cost* $\mathcal{A}(T, u)$ of this scheme, run on tree T from the starting node u, is the worst-case number of edge traversals taken over all of the above mentioned choices of an adversary.

The competitive ratio of $\mathcal{A}$ is defined as

$$\mathcal{R}(\mathcal{A}) = \sup_{T,u} \frac{\mathcal{A}(T,u)}{opt(T,u)} ,$$

where the supremum is taken over all trees T and all starting nodes u of T.

The fact that an oracle is defined on unlabeled rather than labeled trees is an important distinction. For example, for the class of lines, we will prove that an oracle of (asymptotic) size $\log \log n$ is needed to achieve competitive ratio smaller than 2, where n is the length of the line. However, for a *given* labeling, a single bit (indicating the port at the starting node leading to the closer endpoint of the line) is enough to achieve competitive ratio 1: DFS starting toward the closer endpoint achieves it.

The following remark will be useful for proving lower bounds on the competitive ratio of exploration algorithms. Suppose that the robot, at some point of the exploration, is at node v, then moves along an already explored edge e incident to v, and immediately returns to v. For any set of decisions of an adversary, an algorithm causing such a pair of moves, when run on a tree T from some starting node u, has cost strictly larger than the algorithm that skips these two moves. Hence, we restrict attention to exploration algorithms that never perform such returns. We call them *regular*.

In [14] the authors introduced the following classification of exploration algorithms for the class of lines (they considered exploration algorithms that know the length n of the line). Fix n and let *type* k be the set of algorithms that always do at most k returns before reaching an endpoint, and that do exactly this many returns for some combination of starting node and (adversary) choice of the initial direction. They proved the following result that permits to restrict attention to relatively simple algorithms exploring lines, when looking for minimum competitive ratio.

Lemma 1. [14] *Fix* $n \geq 11$. *For every exploration algorithm* $\mathcal{A}$ *for the line* L_n *of length* n *there exists an algorithm* $\mathcal{A}'$ *for* L_n*, such that* $\mathcal{A}'$ *is of type* 1 *and* $\max_{u \in L_n} \frac{\mathcal{A}'(L_n,u)}{opt(L_n,u)} \leq \max_{u \in L_n} \frac{\mathcal{A}(L_n,u)}{opt(L_n,u)}$.

In our setting, an algorithm does not know the length of the line but only the value of the oracle. Hence we change the notion of type in the following way. Consider an algorithm $\mathcal{A}$ using oracle $\mathcal{O}$. Fix a string s in the range of $\mathcal{O}$ and consider the exploration scheme produced by $\mathcal{A}$ for this string. This scheme is of type k if it always does at most k returns before reaching an endpoint, for any line L_n of length n compatible with s, and any starting node u, and if it does exactly this many returns for some line compatible with s, some starting node and some adversary choice of the initial direction.

In the proof of Lemma 1, the algorithm $\mathcal{A}'$ is obtained from $\mathcal{A}$ independently of n. Hence this lemma implies that in our setting the best competitive ratio for the class of lines is achieved by an exploration algorithm that, for any string s, produces a scheme of type 1. This type consists of simple exploration schemes that go x steps in one direction (unless an endpoint is met), then return and go to an endpoint, then return and go to the other endpoint. For any scheme of type 1, this integer x will be called the *probing distance* of the scheme.

The next lemma describes the performance of schemes of type 1 as a function of the probing distance. The proof of the lemma will appear in the full version of the paper.

Lemma 2. *For any positive integer n and any $\alpha < 1$, let $S_{\alpha,n}$ be the exploration scheme of type 1 for the line L_n of length n, with probing distance $\lfloor \alpha n \rfloor$, and let $t_{\alpha,n}(u)$ be the cost of this scheme for starting node u. Let $F_n(\alpha) = \max_{u \in L_n} \frac{t_{\alpha,n}(u)}{opt(L_n,u)}$. Then, there exists a positive integer N_0, such that for any $n \geq N_0$, the function F_n is strictly decreasing in the interval $(0, \frac{\sqrt{3}-1}{2}]$, and $\sup_{n>0} F_n(\alpha) < 2$, for any α in this interval.*

3 The Upper Bound

In this and the next section, we prove our main result, establishing the exact threshold on the size of an oracle for which an exploration algorithm can have competitive ratio smaller than 2. This result is presented in two theorems, one of which establishes an upper bound on the size of such an oracle, by constructing an appropriate exploration algorithm, and the other, in section 4, proves a matching lower bound. In this section, we establish the upper bound, by constructing exploration algorithm `SKE`(c) (for Small-Knowledge-Exploration(c)), for an arbitrary positive integer constant c. This algorithm has competitive ratio smaller than 2, and uses an oracle $\mathcal{O}_c$ of size at most $\max(1, \log\log D - c)$, for any tree of diameter D.

We first describe the oracle $\mathcal{O}_c$. Fix $c > 0$. Given a tree T of diameter D, the oracle $\mathcal{O}_c$ outputs a bit called `choice` and, if `choice` $= 1$, an integer k using $\lceil \log \lceil \log D \rceil \rceil - (c+3)$ bits. The bit `choice` is used by the algorithm to make a decision concerning two alternative ways of exploration, and the integer k is used to obtain an approximation D_0 of the diameter.

Let N_0 be an integer (whose existence is guaranteed by Lemma 2) such that, for all $n \geq N_0$, the function F_n is strictly decreasing in the interval $(0, \frac{\sqrt{3}-1}{2}]$, and $\sup_{n>0} F_n(\alpha) < 2$, for any α in this interval. For $\alpha \in (0, \frac{\sqrt{3}-1}{2}]$, let $\beta(\alpha) = \sup_{n>0} F_n(\alpha)$. Let T be any tree and let n and D be, respectively, its number of nodes and its diameter. Take ϵ such that $D = (1-\epsilon)n$. We will use the following abbreviations: $\lambda = \frac{\sqrt{3}-1}{2}$, and $\gamma = 2^{2^{c+3}+1}$. We now define a threshold ϵ^* on the value of ϵ that will serve to define the bit `choice`. Let $\epsilon_1 = \frac{\lambda}{16\gamma}$, $\beta_1 = \beta(\epsilon_1)$, $\epsilon_2 = \frac{2-\beta_1}{624}$, and $\epsilon^* = \min(\epsilon_1, \epsilon_2)$. The oracle sets `choice` to 1 if

$$(\epsilon < \epsilon^*) \wedge (D \geq 2^{2^{c+3}}) \wedge (n \geq N_0) \ ,$$

and sets `choice` to 0 otherwise. If `choice` $= 1$, the oracle computes $k = \lfloor \frac{\lceil \log D \rceil}{2^{c+3}} \rfloor$.

Given `choice` and k, Algorithm `SKE`(c) returns an exploration scheme. If `choice` $= 0$, then this scheme is an arbitrary DFS. To fix attention, we take the DFS that always chooses the smallest yet unused port number at every node. Note that `choice` is set to 0 when the diameter of the tree is significantly smaller than its size, or when the diameter is bounded, or when the tree itself is small.

We now describe the much more subtle scheme $\mathcal{X}_c$ produced by the algorithm when `choice` = 1. The scheme $\mathcal{X}_c$ uses Procedure DPDFS(v) (for Doubling-Partial-Depth-First-Search(v)) that is called at a node v of the explored tree, outputs the two edges connecting v to the two largest subtrees rooted at neighbors of v, completely explores all other subtrees, and eventually returns to v. In the sequel, we will use the notion of a subtree pending from v as an equivalent to the notion of a subtree rooted at a neighbor of v. Procedure DPDFS(v) is described in Figure 1.

```
Procedure DPDFS($v$)
    $i \leftarrow 1$;
    $S \leftarrow$ set of edges incident to $v$, connecting $v$ to subtrees
          not yet completely explored;
    while $|S| \geq 3$ do
        $S' \leftarrow S$;
        while $S' \neq \emptyset$ do
            let $e \in S'$ and let $T(e)$ be the subtree connected to $v$ by edge $e$;
            explore $T(e)$ by DFS until $\min(|T(e)|, 2^i - 1)$ nodes are visited;
            return to $v$;
            $S' \leftarrow S' \setminus \{e\}$;
            if $T(e)$ is completely explored then $S \leftarrow S \setminus \{e\}$;
        $i \leftarrow i + 1$;
    if $|S| = 2$ then return $S$;
    if $|S| = 1$ then let $e'$ be the edge connecting $v$ to the largest
          explored subtree and return $S \cup \{e'\}$;
    if $S = \emptyset$ then let $e'$ and $e''$ be the edges connecting $v$ to the two largest
          explored subtrees and return $\{e', e''\}$;
```

Fig. 1. Procedure DPDFS

The proof of the following lemma will appear in the full version of the paper.

Lemma 3. *Let v be any node of degree at least 3. Let $T_1, \ldots, T_p$ be the enumeration of the subtrees pending from v in decreasing order of their sizes. Procedure* DPDFS(v) *returns two edges corresponding to two largest subtrees (up to size equality), and completely explores all other subtrees pending from v. Moreover, the cost of Procedure* DPDFS(v) *is at most* $22 \sum_{i \geq 3}^{p} |T_i|$.

The intuitive idea of the exploration scheme $\mathcal{X}_c$ (returned by Algorithm SKE(c) when `choice` = 1) is the following. Let $D_0 = 2^{k \cdot 2^{c+3} - 1}$. We will prove that D_0 approximates the diameter D as follows: $D_0 \leq D < \gamma D_0$. The robot uses Procedure DPDFS(v) to identify the two edges connecting the current node v to the largest subtrees pending from it. Then the robot moves along one of the edges and applies the procedure again. These consecutive applications define a path of length approximately equal to the diameter of the tree. On this path the robot applies a scheme of type 1 for lines: go at probing distance $\lfloor \lambda D_0/2 \rfloor$, return and go to the endpoint of the path, return and go to the other endpoint of the path. The approximation D_0 of the diameter is tight enough to guarantee good performance of the scheme on this path. On the other hand, the part of the tree disjoint from this path is negligible (this is implied by the conditions of setting `choice` to 1). These two facts (shown in detail in the proof of Theorem 1) imply that the competitive ratio of scheme $\mathcal{X}_c$ is smaller than 2.

The description of the exploration scheme $\mathcal{X}_c$ is provided in Figure 2. In the description, moves performed during the calls to Procedure `DPDFS` are called *internal*, and all other moves are called *external*. During the entire exploration, the robot stores the results of all previous actions, and constructs a map of the portion of the tree that has been explored so far.

The proof of the following lemma will appear in the full version of the paper.

Lemma 4. *Algorithm* `SKE`(c) *is correct.*

Theorem 1. *Let c be an arbitrary positive integer constant. Algorithm* `SKE`(c) *uses an oracle of size at most* $\max(1, \log\log D - c)$*, for any tree of diameter D, and has competitive ratio smaller than 2.*

Proof. The result is true for $n = 1$ or $n = 2$, as any algorithm is optimal in this case. In the following, we assume that $n \geq 3$.

Recall that $k = \lfloor \frac{\lceil \log D \rceil}{2^{c+3}} \rfloor$. The oracle $\mathcal{O}_c$ uses at most $\lceil \log k \rceil + 1$ bits, hence at most $\max(1, \log\log D - c)$ bits. The definition of k implies the inequality $k \leq \frac{\lceil \log D \rceil}{2^{c+3}} < k + 1$, hence $k \cdot 2^{c+3} \leq \lceil \log D \rceil < k \cdot 2^{c+3} + 2^{c+3}$, and finally $2^{k \cdot 2^{c+3} - 1} \leq D < 2^{k \cdot 2^{c+3}} \cdot 2^{2^{c+3}}$. From the definition of D_0 and γ we get $D_0 \leq D < \gamma D_0$.

First assume that the oracle sets `choice` to 0. Since the exploration scheme in this case is a DFS, the cost of the scheme is at most $2(n-1) - 1 = 2n - 3$. The cost $opt(T, u)$, where u is the starting node, is $2(n-1) - ecc(u)$. We have $ecc(u) \leq D = (1 - \epsilon)n$. We obtain

$$opt(T, u) \geq 2(n-1) - (1-\epsilon)n = (1+\epsilon)n - 2 \ .$$

Since $D \leq n - 1$, we have $\epsilon \geq 1/n$, or equivalently $\epsilon n \geq 1$. Hence the ratio in this case is at most

$$\frac{2n-3}{(1+\epsilon)n - 2} = \frac{2n-3}{(1+\epsilon/2)n - 2 + \epsilon n/2} \leq \frac{2n-3}{(1+\epsilon/2)n - 1.5} \leq \frac{2}{1+\epsilon/2} \ .$$

If $\epsilon \geq \epsilon^*$, then $\frac{2}{1+\epsilon/2} \leq \frac{2}{1+\epsilon^*/2} < 2$. Assume that $\epsilon < \epsilon^*$. Let $D^* = 2^{2^{c+3}}$. Therefore, `choice` is set to 0 because either $D < D^*$ or $n < N_0$. If $D < D^*$, then $n < \frac{D^*}{1-\epsilon}$. Let $N_1 = \frac{D^*}{1-\epsilon^*}$. Then we have $n < \frac{D^*}{1-\epsilon} \leq \frac{D^*}{1-\epsilon^*} = N_1$. Hence,

```
Exploration scheme $\mathcal{X}_c$
    while the exploration is not completed do
        let $v$ be the current node;
        {Internal moves:}
        if $\deg(v) \geq 3$ then
            unless Procedure DPDFS($v$) has already been applied in a previous
            step, apply it to get edges $e, e'$ connecting $v$ to the two largest
            subtrees pending from $v$;
        {External move:}
        if there is only one edge connecting $v$ to a subtree not completely
        explored then
            leave $v$ by this edge;
        else
            {$e, e'$ are the two edges connecting $v$ to the two subtrees
            not yet completely explored}
            if during the last external move (if any), the robot did not
            come to $v$ by $e$ or $e'$ then
                leave $v$ by edge $e$;
            else
                assume w.l.o.g. that the robot came to $v$ by edge $e$;
                if the robot is for the first time at distance $\lfloor \lambda D_0/2 \rfloor$ from the
                starting node then
                    leave $v$ by edge $e$;
                else leave $v$ by edge $e'$;
    endwhile
```

Fig. 2. Exploration scheme $\mathcal{X}_c$

both when $D < D^*$ and when $n < N_0$, we have $n < N^* = \max(N_0, N_1)$. Let $\epsilon_3 = 1/N^*$. We have $\epsilon \geq 1/n \geq 1/N^* = \epsilon_3$. We obtain $\frac{2}{1+\epsilon/2} \leq \frac{2}{1+\epsilon_3/2} < 2$. Hence the ratio of the cost of DFS (returned by Algorithm `SKE`(c) when `choice` is set to 0) to $opt(T, u)$, is at most $\max(\frac{2}{1+\epsilon^*/2}, \frac{2}{1+\epsilon_3/2}) < 2$.

From now on, we assume that the oracle sets `choice` to 1, hence Algorithm `SKE`(c) returns exploration scheme $\mathcal{X}_c$.

In the analysis of the cost of exploration scheme $\mathcal{X}_c$, we use the following terminology. Assume that the robot enters some node of degree at least 3 by edge e and applies Procedure `DPDFS`(v). If the procedure outputs two edges different from e, then we say that the current node v is a *fork*. Now consider edges traversed during external moves. These edges form a subtree T' of T. For any node v, there exist at most two incident edges such that any external move of the robot leaving v takes one of them. Hence, all nodes are of maximal degree 3 in this subtree. Nodes of degree exactly 3 in T' are forks. Let $v_1, \ldots, v_q$ be the forks of T', if any, in order of their first visit by the robot. Let e_i be the edge connecting v_i to the subtree pending from it and containing the starting node u. In view of the definition of a fork, the robot never makes an external move on edge e_i from node v_i. Let u' be the last fork v_q, if any, or $u' = u$, if T' does not contain any fork. Finally, let P be a path of length D in the tree and let P' be the set of nodes in T' visited by an external move after u'. P' is a path because it does not contain any fork other than possibly u'. Finally, let C be the length of a shortest covering walk in path P' starting at node u'.

In our analysis, the cost of the scheme $\mathcal{X}_c$ is split into the cost of internal moves, and the cost of external moves. The proofs of the following claims will appear in the full version of the paper.

Claim 1. The total number of internal moves is at most $154\epsilon n$.

Claim 2. The total number of external moves is at most $\beta_1 C + 2\epsilon n$.

Claim 1 and Claim 2 imply that the total cost of the scheme $\mathcal{X}_c$ is at most $\beta_1 \cdot C + (154+2)\epsilon n = \beta_1 \cdot C + 156\epsilon n$.

It remains to bound the ratio ρ of this cost to $opt(T,u)$. The shortest covering walk starting at u visits u' before any other node of P'. It has then to visit the path P' starting from u'. Therefore, the length of the shortest covering walk starting at u cannot be less than C (the optimal number of moves on P' starting from u'). This gives $\rho \leq \frac{\beta_1 \cdot C + 156\epsilon n}{C}$. We have $\epsilon \leq \epsilon_2 = \frac{2-\beta_1}{624}$. Together with $C \geq |P'| \geq n/2$ (for the latter inequality, see the proof of Claim 2), this implies

$$\beta_1 + \frac{156\epsilon n}{C} \leq \beta_1 + \frac{156\epsilon n}{n/2} \leq \beta_1 + 2 \cdot 156 \frac{2-\beta_1}{624} = \beta_1 + \frac{2-\beta_1}{2} = 1 + \frac{\beta_1}{2} < 2 \ .$$

It follows from the above obtained estimates that the competitive ratio of Algorithm SKE(c) is at most

$$\max\left(\frac{2}{1+\epsilon^*/2}, \frac{2}{1+\epsilon_3/2}, 1+\frac{\beta_1}{2}\right) < 2 \ ,$$

which completes the proof of the theorem. □

4 The Lower Bound

This section is devoted to establishing a lower bound on the size of an oracle for which there exists an algorithm with competitive ratio smaller than 2. This lower bound exactly matches the upper bound shown previously, and it holds even for the class of lines. Indeed, we show that for oracles whose size for all lines L_k, of diameter (i.e., length) $k \leq n$, is smaller than $\log\log n$, and differs from it by an unbounded number of bits, every algorithm has competitive ratio at least 2.

Theorem 2. *Let $\mathcal{O}$ be an oracle and let $f(n)$ denote the maximum of sizes of $\mathcal{O}(L_k)$, for $k \leq n$. Let $g : \mathbb{N} \mapsto \mathbb{R}$ be defined by the formula $f(n) = \log\log n - g(n)$. If g is a function unbounded from above, then every exploration algorithm using oracle $\mathcal{O}$ has competitive ratio at least 2.*

Proof. We will use the following claim.

Claim 3. For every positive integers M and γ, there exist integers $n_1 > n_2 \geq M$, such that $\mathcal{O}(L_{n_1}) = \mathcal{O}(L_{n_2})$ and $n_1/n_2 \geq \gamma$.

Suppose that the claim does not hold. Take M and γ that refute it. Let $\psi : \{n : n > M\} \mapsto I\!R$ be the sequence defined by the formula $\psi(n) = \frac{\log \gamma \log n}{\log n - \log M}$. The sequence ψ converges to $\log \gamma$, hence it is bounded. Let A be such that $\psi(n) < A$ for all n. Since g is an unbounded function, there exists $n_0 > M$ for which $g(n_0) > \log A$. Let x be the size of the set $\{\mathcal{O}(L_k) : k \leq n_0\}$. We have

$$x \leq 2^{f(n_0)} = 2^{\log\log n_0 - g(n_0)} < 2^{\log\log n_0 - \log A} < 2^{\log\log n_0 - \log \frac{\log \gamma \log n_0}{\log n_0 - \log M}}$$

and therefore $x < \frac{\log n_0 - \log M}{\log \gamma}$. All integers k with $\mathcal{O}(L_k) = \mathcal{O}(L_{M\gamma^i})$ must be smaller than $M\gamma^{i+1}$, for $i \geq 0$. Hence all oracle values for lines $L_{M\gamma^i}$ are distinct, and there are x such values. We have $n_0 < M\gamma^x$ because $\mathcal{O}(L_{n_0}) = \mathcal{O}(L_{M\gamma^i})$, for some $i < x$, and hence $n_0 < M\gamma^{i+1} \leq M\gamma^x$. Consequently, $\log n_0 \leq \log M + x \log \gamma < \log M + \log n_0 - \log M$. This contradiction proves Claim 3.

We will now show that any algorithm using oracle $\mathcal{O}$ must have competitive ratio at least 2. In view of Lemma 1 it is enough to restrict attention to algorithms producing exploration schemes of type 1 for the class of lines. The probing distance of such a scheme for line L_n depends only on $\mathcal{O}(L_n)$. Consider an algorithm $\mathcal{A}$ producing a scheme of type 1 with probing distance $\phi(\mathcal{O}(L_n))$. Fix any constant $3/2 < \beta < 2$. Choose γ such that $\frac{2\gamma}{\gamma+2} > \beta$ and M such that $\frac{2M-1}{M+1} > \beta$. Hence $\gamma > 6$. Let $n_1 > n_2 \geq M$ be integers for which $\mathcal{O}(L_{n_1}) = \mathcal{O}(L_{n_2})$ and $n_1 \geq \gamma n_2$. Their existence is guaranteed by Claim 3. Let $y = \phi(\mathcal{O}(L_{n_1}))$. Hence the scheme makes the first change of direction after y steps, both in L_{n_1} and in L_{n_2}, unless an endpoint is encountered earlier. Consider two cases.

If $y \leq n_2$ then consider the behavior of $\mathcal{A}$ on L_{n_1}, with the starting node u at distance $y+1$ from the endpoint toward which the robot starts. Since $\gamma > 6$, this is the endpoint closer to u. Then

$$\frac{\mathcal{A}(L_{n_1}, u)}{opt(L_{n_1}, u)} = \frac{y + 2n_1 - 1}{y + n_1 + 1} \geq \frac{n_2 + 2n_1 - 1}{n_2 + n_1 + 1}$$
$$\geq \frac{(2\gamma+1)n_2 - 1}{(\gamma+1)n_2 + 1} \geq \frac{(2\gamma+1) - 1}{(\gamma+1) + 1} = \frac{2\gamma}{\gamma+2} > \beta \ .$$

If $y > n_2$ then consider the behavior of $\mathcal{A}$ on L_{n_2} with the starting node u at distance $n_2 - 1$ from the endpoint toward which the robot starts. Then

$$\frac{\mathcal{A}(L_{n_2}, u)}{opt(L_{n_2}, u)} = \frac{2n_2 - 1}{n_2 + 1} \geq \frac{2M - 1}{M + 1} > \beta \ .$$

This proves that the competitive ratio of algorithm $\mathcal{A}$ is at least 2. □

5 Exploration Knowing the Diameter

We have shown in Section 3 that very little information (less than $\log\log D$ bits) is needed to beat competitive ratio 2, and in fact, most of this information (all bits except one) concerns the value of the diameter D itself, and is used to establish a lower bound on it. This extra bit, however, cannot be deduced from

D alone, and turns out to be crucial. In this section we prove a surprising result that even an algorithm that knows D *exactly* (i.e., is provided with all $\lceil \log D \rceil$ bits of it), but does not have any additional knowledge, cannot beat competitive ratio 2. Notice that a similar argument proves that the exact knowledge of the number n of nodes, with no extra information, is not enough for this purpose either. The proof of the following theorem will appear in the full version of the paper.

Theorem 3. *Let $\mathcal{A}$ be any tree exploration algorithm that, for every tree T, is given the diameter of T as input. Then $\mathcal{A}$ has competitive ratio at least 2.*

Acknowledgment. Pierre Fraigniaud and David Ilcinkas were partially supported by the projects PairAPair of the ACI Masses de Données, and FRAGILE of the ACI Sécurité Informatique. Additional support from the INRIA project "Grand Large".

Andrzej Pelc was partially supported by NSERC discovery grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais. This work was done during the visit of Andrzej Pelc at LRI. Additional supports from the Ministère des Relations Internationales du Québec, from University of Paris-Sud, and from the project PairAPair of the ACI Masses de Données.

References

1. S. Albers and M. R. Henzinger, Exploring unknown environments, SIAM Journal on Computing 29 (2000), 1164-1188.
2. B. Awerbuch, M. Betke, R. Rivest and M. Singh, Piecemeal graph learning by a mobile robot, Proc. 8th Conf. on Comput. Learning Theory (1995), 321-328.
3. E. Bar-Eli, P. Berman, A. Fiat and R. Yan, On-line navigation in a room, Journal of Algorithms 17 (1994), 319-341.
4. M.A. Bender, A. Fernandez, D. Ron, A. Sahai and S. Vadhan, The power of a pebble: Exploring and mapping directed graphs, Information and Computation 176 (2002), 1-21.
5. M.A. Bender and D. Slonim, The power of team exploration: Two robots can learn unlabeled directed graphs, Proc. 35th Ann. Symp. on Foundations of Computer Science (FOCS 1994), 75-85.
6. P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen and M. Saks, Randomized robot navigation algorithms, Proc. 7th ACM-SIAM Symp. on Discrete Algorithms (SODA 1996), 74-84.
7. A. Blum, P. Raghavan and B. Schieber, Navigating in unfamiliar geometric terrain, SIAM Journal on Computing 26 (1997), 110-137.
8. M. Betke, R. Rivest and M. Singh, Piecemeal learning of an unknown environment, Machine Learning 18 (1995), 231-254.
9. A.E.F. Clementi, A. Monti and R. Silvestri, Selective families, superimposed codes, and broadcasting on unknown radio networks, Proc. 12th Ann. ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), 709-718.
10. A. Czumaj and W. Rytter, Broadcasting algorithms in radio networks with unknown topology, Proc. 44th Ann. Symposium on Foundations of Computer Science (FOCS 2003), 492-501.

11. X. Deng, T. Kameda and C. H. Papadimitriou, How to learn an unknown environment I: the rectilinear case, Journal of the ACM 45 (1998), 215-245.
12. X. Deng and A. Mirzaian, Competitive robot mapping with homogeneous markers, IEEE Transactions on Robotics and Automation 12 (1996), 532-542.
13. X. Deng and C. H. Papadimitriou, Exploring an unknown graph, Journal of Graph Theory 32 (1999), 265-297.
14. A. Dessmark and A. Pelc, Optimal graph exploration without good maps, Theoretical Computer Science 326 (2004), 343-362.
15. K. Diks, P. Fraigniaud, E. Kranakis and A. Pelc, Tree exploration with little memory, Journal of Algorithms 51 (2004), 38-63.
16. G. Dudek, M. Jenkin, E. Milios and D. Wilkes, Robotic exploration as graph construction, IEEE Transactions on Robotics and Automation 7 (1991), 859-865.
17. V. Dujmović and S. Whitesides, On validating planar worlds, Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), 791-792.
18. C.A. Duncan, S.G. Kobourov and V.S.A. Kumar, Optimal constrained graph exploration, Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2001), 807-814.
19. F. Fich and E. Ruppert. Hundreds of impossibility results for distributed computing, Distributed Computing, 16 (2003), 121–163.
20. R. Fleischer and G. Trippen, Exploring an unknown graph efficiently, Proc. 13th Ann. European Symposium on Algorithms (ESA 2005), LNCS 3669, 11-22.
21. P. Fraigniaud, D. Ilcinkas and A. Pelc. Oracle size: a new measure of difficulty for communication tasks, Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC 2006), to appear.
22. L. Gasieniec, D. Peleg and Q. Xin, Faster communication in known topology radio networks, Proc. 24th Annual ACM Symposium on Principles of Distributed Computing (PODC 2005), 129-137.
23. T. Kameda and M. Yamashita. Computing on anonymous networks: Part I – characterizing the solvable cases. IEEE Transactions on Parallel and Distributed Systems, 7 (1996), 69-89.
24. D. Kowalski and A. Pelc, Optimal deterministic broadcasting in known topology radio networks, manuscript.
25. N. Lynch. A hundred impossibility proofs for distributed computing. Proc. 8th Ann. ACM Symposium on Principles of Distributed Computing (PODC 1989),1-28.
26. P. Panaite and A. Pelc, Exploring unknown undirected graphs, Journal of Algorithms 33 (1999), 281-295.
27. P. Panaite and A. Pelc, Impact of topographic information on graph exploration efficiency, Networks, 36 (2000), 96-103.
28. C. H. Papadimitriou and M. Yannakakis, Shortest paths without a map, Theoretical Computer Science 84 (1991), 127-150.

Distributed Data Structures: A Survey on Informative Labeling Schemes

Cyril Gavoille[⋆]

LaBRI, Université Bordeaux 1
351, Cours de la Libération,
33405 Talence, France
gavoille@labri.fr

Abstract. In this talk, we will survey the role of data structures for compactly storing and representing various types of information in a localized and distributed fashion. Traditional approaches to data representation are based on global data structures, which require access to the entire structure even if the sought information involves only a small and local set of entities. In contrast, localized data representation schemes are based on breaking the information into small local pieces, or *labels*, selected in a way that allows one to infer information regarding a small set of entities directly from their labels, without using any additional (global) information.

Keywords: data-structures, distributed algorithm, labeling scheme, graphs.

[⋆] Supported by the project "PairAPair" of the ACI Masses de Données.

From Deduction Graphs to Proof Nets: Boxes and Sharing in the Graphical Presentation of Deductions

Herman Geuvers and Iris Loeb

Radboud University Nijmegen, Institute for Computing and Information Sciences,
6500 GL Nijmegen, The Netherlands
{H.Geuvers, I.Loeb}@cs.ru.nl

Abstract. Deduction graphs [3] provide a formalism for natural deduction, where the deductions have the structure of acyclic directed graphs with boxes. The boxes are used to restrict the scope of local assumptions. Proof nets for multiplicative exponential linear logic (MELL) are also graphs with boxes, but in MELL the boxes have the purpose of controlling the modal operator !. In this paper we study the apparent correspondences between deduction graphs and proof nets, both by looking at the structure of the proofs themselves and at the process of cut-elimination defined on them. We give two translations from deduction graphs for minimal proposition logic to proof nets: a direct one, and a mapping via so-called context nets. Context nets are closer to natural deduction than proof nets, as they have both premises (on top of the net) and conclusions (at the bottom). Although the two translations give basically the same results, the translation via context nets provides a more abstract view and has the advantage that it follows the same inductive construction as the deduction graphs. The translations behave nicely with respect to cut-elimination.

1 Introduction

Deduction graphs [3] provide a formalism for natural deduction where the deductions have the structure of an *acyclic directed graph* with *boxes*. The main advantage of this formalism is the re-use of sub-proofs, which is simply done by putting several arrows to its conclusion. Because we do not see this as a logical step, we call this kind of sharing *implicit*. This makes deduction graphs a generalization of both Gentzen-Prawitz style natural deduction and Fitch style natural deduction. Because of the graph structure, one has to be explicit about local assumptions, so the boxes are used to limit the scope of an assumption.

In proof nets for multiplicative exponential linear logic (MELL) [4], the re-use of formulas is done by contraction links, and is therefore *explicit*. Proof nets also have boxes. Here they sequentialise the proof net in order to introduce the global modality "!" (of course).

There are some well-known translations from simply typed λ-terms to proof nets. There are three reasons why a translation from deduction graphs to proof nets is more difficult. Firstly, parts of a deduction graph can be shared. We want a translation to somehow reflect this sharing. Simply typed λ-terms do not have sharing, except for the variables. Secondly, deduction graphs contain boxes. We would like a translation to associate a proof net box to a deduction graph box. Furthermore, the sharing and the boxes are both affected during the process of cut-elimination on deduction graphs. We want the cut-elimination on deduction graphs to behave similarly to cut-elimination on proof nets (on the translated deduction graphs). Thirdly, deduction graphs may have many conclusions, unlike simply typed terms, which have just one type. In this paper we present two translations from deduction graphs to proof nets, which mainly differ in the way they handle multiple conclusions.

We start by presenting *deduction graphs with explicit sharing*, SG, which is a variant of the usual definition of deduction graphs. The main difference is that in SG, sharing is handled as a logical step (Section 2). Furthermore we add a step, "loop", that allows to distinguish between conclusions and garbage, that is between formulas that we can use as the premises of next steps and formulas that will not be used any more.

The translation $(-)^*$ of Girard is used to translate formulas of minimal proposition logic to formulas of MELL (Section 3). This translation is the most suitable to our needs because it allows to nicely map deduction graph boxes to proof net boxes.

The direct translation (Section 4) uses *tensors* ($\otimes$) to connect the various conclusions. Just as most translations from λ-calculus to proof nets, assumptions Γ occur in the translation as terminal nodes $\Gamma^{*\perp}$, so as the *negation* of the translated formulas. An advantage of the direct translation is, that it only uses concepts that are already known from the theory of proof nets. A disadvantage is, that it works from conclusions to assumptions. So to translate a deduction graph, we must know that the construction of the graph has been finished. If we would decide to extend the graph later, we would have to translate the whole graph again.

Context nets (Section 5) form an extension of the concept of proof nets. They have terminal nodes, but they also permit so called *initial nodes.* We argue that there are several sensible translations from deduction graphs to proof nets, the translation of Section 4 being one of them, and that the translation to context nets generalises them by investigating the common parts of those proof nets. The translation from deduction graphs to context nets also reveals a symmetry between assumptions and conclusions: not only do the assumptions Γ associate to terminal nodes $\Gamma^{*\perp}$, but the conclusions Δ associate to initial nodes $\Delta^{*\perp}$ as well. Moreover, the translation can be done in the same order as the construction of the deduction graph, allowing to work with graphs that are not yet finished. In fact, the construction of the translation of a deduction graph looks like the construction of the original deduction graph "up side down".

In Section 6 it is shown that the translation via context nets after a slight modification, yields the same results as the direct translation. Section 7, finally, shows that the translations behave well with respect to cut-elimination.

2 Deduction Graphs with Explicit Sharing

In [3], we have defined the notion of *deduction graph* (DG), which is a generalization of both Gentzen-Prawitz style natural deduction and Fitch-style flag-deduction. A deduction graph is an acyclic directed graph with boxes satisfying some conditions. The boxes take care of local assumptions that can be *discharged*: a box restricts the scope of nodes and a box is a node itself. The nodes in the graph are labelled with formulas and if, e.g. node k is labelled with $A{\rightarrow}B$ and node m is labelled with A, there can be a node n labelled with B with one edge pointing to $(k, A{\rightarrow}B)$ and one pointing to (m, A). The general idea is that the inverse of the edges represents logical derivability: if from node n with label A there are edges to nodes $n_1, \ldots, n_k$ with labels $A_1, \ldots, A_k$ then A should be derivable from $A_1, \ldots, A_k$ using a logical rule. In [3], an inductive definition of the notion of *deduction graph for minimal propositional logic* is given. We do not repeat the definition here, but give as an example the left graph in Figure 1, which should also clarify the use of boxes.

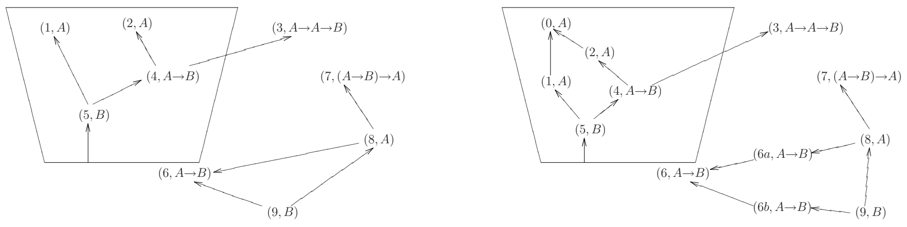


Fig. 1. Example of a deduction graph (left) and the same graph with an explicit sharing construction (right)

In the figure, node 6 is the node of the *box* containing nodes $1, 2, 4, 5$. The idea is that a set of nodes $\mathcal{B}$ (a box) can form a node again, which is called the *box node* of $\mathcal{B}$. So 6 is the box node of $\{1, 2, 4, 5\}$. The nodes that are not inside a box are on the *top level* and the top level nodes without incoming edges are called *free nodes*. So 3 and 7 are the free nodes in the example. Their labels $A{\rightarrow}A{\rightarrow}B$ and $(A{\rightarrow}B){\rightarrow}A$ correspond to the assumptions of the deduction. We can view this deduction graph as a deduction of B from $A{\rightarrow}A{\rightarrow}B$ and $(A{\rightarrow}B){\rightarrow}A$. The nodes 1 and 2 have no incoming edges but are inside a box. These are the local assumptions, which are discharged when forming the box. In deduction graphs it is not allowed to have edges pointing into a box, because this would correspond to the global use of a local assumption. Also overlapping boxes are not allowed (but boxes may be contained in each other). This is taken care of by the requirement that deduction graphs should be *closed box directed graphs*.

All definitions regarding deduction graphs are in [3], but because it is quite important in this paper we repeat the notion of closed box directed graph here:

Definition 1. closed box directed graph *is a triple* $\langle X, G, (\mathcal{B}_i)_{i \in I}\rangle$ *where* G *is a directed graph where all nodes have a* label *in* X *and* $(\mathcal{B}_i)_{i \in I}$ *is a collection of* *sets of nodes of G, the* boxes. *Each box $\mathcal{B}_i$ corresponds to a node, the* box node *of $\mathcal{B}_i$. Moreover, the boxes $(\mathcal{B}_i)_{i\in I}$ satisfy the following properties.*

1. *(Non-overlap) Two boxes are disjoint or one is contained in the other: $\forall i, j \in I(\mathcal{B}_i \cap \mathcal{B}_j = \emptyset \vee \mathcal{B}_i \subset \mathcal{B}_j \vee \mathcal{B}_j \subset \mathcal{B}_i)$,*
2. *(Box-node edge) There is only one outgoing edge from a box-node and that points into the box itself (i.e. to a node in the box),*
3. *(No edges into a box) Apart from the edge from the box-node, there are no edges pointing into a box.*

We first redefine the notion of deduction graph (of [3]) a bit, where we make the sharing explicit via a *sharing construction* (`SG`). If we have a *conclusion node* (i.e. a top-level node without incoming edges) (n, A), then we can add two nodes (m, A) and (k, A), which both have one outgoing edge to (n, A).

Definition 2. *The collection of* deduction graphs with explicit sharing *(`SG`) is the set of closed box directed graphs over $\mathbb{N} \times \mathsf{Form}$ inductively defined as follows.*

Axiom *A single node (n, A) is an `SG`,*

Join *If G and G' are disjoint `SG`s, then $G'' := G \cup G'$ is an `SG`.*

$\to$-E *If G is an `SG` containing two conclusion nodes $(n, A\to B)$ and (m, A), then the graph $G' := G$ with*
- *a new node (p, B) at the top level*
- *an edge $(p, B) \longrightarrow\!\triangleright (n, A\to B)$,*
- *an edge $(p, B) \longrightarrow\!\triangleright (m, A)$,*

is an `SG`.

Repeat *If G is an `SG` containing a conclusion node (n, A), the graph $G' := G$ with*
- *a new node (m, A) at the top level,*
- *an edge $(m, A) \longrightarrow\!\triangleright (n, A)$*

is an `SG`.

Share *If G is an `SG` containing a conclusion node (n, A), the graph $G' := G$ with*
- *two new nodes (m, A), (k, A) at top level,*
- *an edge $(m, A) \longrightarrow\!\triangleright (n, A)$,*
- *an edge $(k, A) \longrightarrow\!\triangleright (n, A)$*

is an `SG`.

$\to$-I *If G is an `SG` containing a conclusion node (j, B) and a free node (m, A) then the graph $G' := G$ with*
- *A box $\mathcal{B}$ with box-node $(n, A\to B)$, containing the node (j, B) as the only conclusion node, and (m, A), and no other nodes that were free in G,*
- *An edge from the box node $(n, A\to B)$ to (j, B)*

is an `SG` under the proviso that it is a well-formed closed box directed graph.

Loop *If G is an `SG` containing the conclusions (m, A) and (n, B) $(m \neq n)$, then the graph $G := G'$ with an edge $m \longrightarrow\!\triangleright m$ is an `SG`.*

Furthermore, in the construction one should always take care that, if $k \longrightarrow\!\triangleright\, l$ and $k \in \mathcal{B}$ and $l \notin \mathcal{B}$, then:

- *if k has been constructed in a* **Repeat**-*step, then k is not in any boxes contained in $\mathcal{B}$.*
- *k has not been constructed in a* **Share**-*step.*

So, apart from the four types of nodes A, I, E and R that we have already distinguished for deduction graphs in [3], we now also have S. A node is of a certain type, if it has been added in the corresponding construction rule of Definition 2. For example, a node is of type E if it has been added in a →-E-step.

Definition 3. *A top-level node (n, A) such that $n \longrightarrow\!\triangleright\, n$ is called a* fake conclusion node*.*

It is easy to turn a deduction graph into a sharing graph, simply by making the implicit sharing in DGs explicit via S-nodes and by adding repeats if necessary. This is indicated in Figure 1. That the changes in sharing, repeating, looping and arrow introduction are not serious, is stated more precisely in the following lemma.

Lemma 4. *If G is an* DG *with free nodes Γ and conclusion nodes Δ, then there exists an* SG *G' with free nodes Γ and conclusion nodes Δ.*

We now modify the process of cut-elimination, because the deduction graphs with explicit sharing are not closed under the rules we defined for DGs. For example, we started the process by eliminating all repeats that separated the I-node of the cut from the E-node. Because the deduction graphs with explicit sharing are required to have only R-nodes that cross the border of at most one box, we now have to postpone the removal of some R-nodes. Therefore the process will transform the SG levelwise. Further, we now also have to handle the new S-nodes. The changes only affect the order in which we make the cut explicit. The elimination of the *safe cut* itself remains the same. A *safe cut* is the situation where we have an E-node where the edge to the →-formula points to an I-node at the same level, so we have an →-introduction followed immediately by an →-elimination. In that case, the cut can be removed by a simple reordering of edges and the removal of some nodes. This is discussed in detail for DGs in [3]; the situation for SGs is slightly simpler.

Definition 5. *A cut in an* SG *G is a subgraph of G consisting of:*

- *A box-node $(n, A \to B)$,*
- *A node (p, B),*
- *A node (m, A),*
- *A sequence of* R *and* S *nodes $(s_0, A \to B), \ldots, (s_i, A \to B)$,*
- *Edges $(p, B) \longrightarrow\!\triangleright\, (s_i, A \to B) \longrightarrow\!\triangleright \ldots \longrightarrow\!\triangleright\, (s_0, A \to B) \longrightarrow\!\triangleright\, (n, A \to B)$,*
- *An edge $(p, B) \longrightarrow\!\triangleright\, (m, A)$.*

The sequence $(p, B) \longrightarrow\!\triangleright (s_i, A \to B) \longrightarrow\!\triangleright \ldots \longrightarrow\!\triangleright (n, A \to B)$ is called the cut-sequence*.*

Definition 6 (Cut hidden by repeats). *Let G be an* SG *and let c be a cut of G and let S be that part of the cut-sequence that is at the same level as $(n, A \to B)$. If s_j is an* R*-node in S, for some $0 \leq j \leq i$, then the repeat-elimination at s_j is obtained by:*

- *When an edge points to s_j, redirect it to s_{j-1} (or to n, if $j = 0$);*
- *Remove s_j.*

Definition 7 (Cut hidden by sharing). *Let G be an* SG *with a cut c that contains a box $\mathcal{B}$ with box-node $(n, A \to C)$ and in-going edges from p_1, p_2. Then the* unsharing of G at nodes n, p_1, p_2 *is obtained by:*

- *removing $\mathcal{B}$,*
- *adding copies $\mathcal{B}'$ and $\mathcal{B}''$ of $\mathcal{B}$ including copies of the nodes reachable within one step,*
- *Connect the copies outside the boxes, to the original nodes. (thus if we had $q \longrightarrow\!\!\triangleright\, m$ with $q \in \mathcal{B}$, $m \notin \mathcal{B}$ and m' is the copy of m connected to $\mathcal{B}'$, and m'' is the copy of m connected to $\mathcal{B}''$, then we add the edges $m' \longrightarrow\!\!\triangleright\, m$ and $m'' \longrightarrow\!\!\triangleright\, m$),*
- *replacing n'' by p_1 and replacing n' by p_2.*

Figure 2 shows the unsharing of the SG in Figure 1.

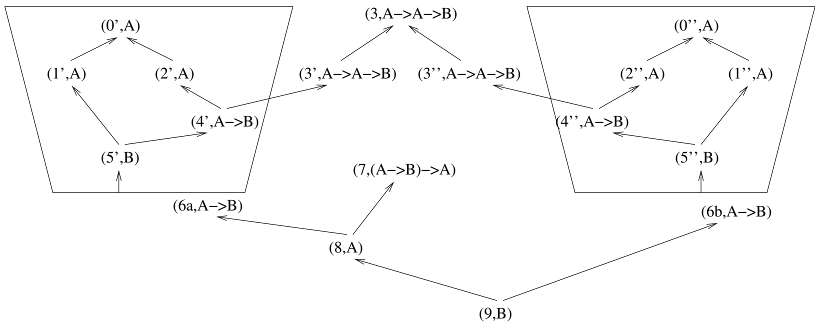


Fig. 2. Unsharing of the SG in Figure 1

Definition 8 (Cut hidden by a depth conflict; incorporation). *We have a* depth conflict *in the* SG *G with cut c, if $(n, A \to B)$ has an incoming edge from a node at another level. Let $\mathcal{B}$ the box of which $(n, A \to B)$ is the box-node. In that case the* incorporation of G at c *is obtained by:*

- *making copies m' of the* R*-nodes m that are reachable from $\mathcal{B}$ within one step,*
- *Replacing edges $q \longrightarrow\!\!\triangleright\, m$ by $q \longrightarrow\!\!\triangleright\, m' \longrightarrow\!\!\triangleright\, m$,*
- *putting both $\mathcal{B}$ and the nodes m' at one level deeper.*

Figure 3 shows an SG with a cut hidden by a depth conflict; Figure 4 shows its incorporation at 7, 12.

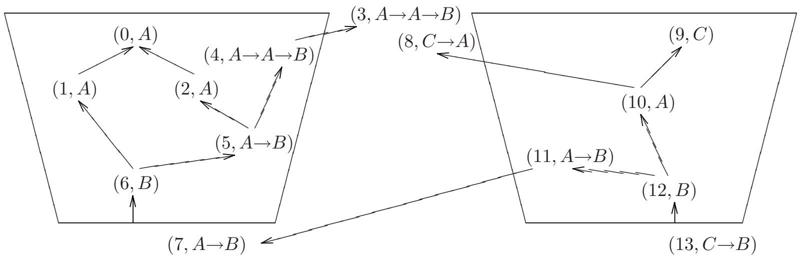


Fig. 3. SG with cut hidden by a depth conflict

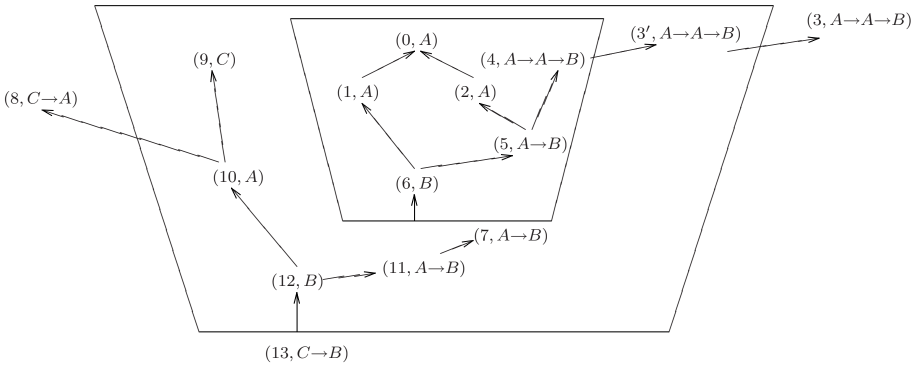


Fig. 4. Incorporation of the SG in Figure 3

Definition 9. *Given an* SG *G with a cut c,* the process of eliminating the cut *c is the following:*

1. *(Repeat-elimination) As often as possible, perform the repeat-elimination step as described in Definition 6.*
2. *(Unsharing) As often as possible, perform the unsharing step as described in Definition 7.*
3. *(Incorporation) If possible, perform the incorporation step as described in Definition 8.*
4. *(Moving up one level) If c is not yet safe, repeat the procedure, starting at 1.*
5. *(Eliminating the safe cut) Eliminate the safe cut.*

The proof of strong normalization of the process of cut-elimination on DGs in [3] makes only use of the fact that it always ends in the elimination of a safe cut; the order in which we apply the other steps –Repeat-elimination, unsharing and incorporation– plays no role. As the process of cut-elimination on SGs still ends in the elimination of a safe cut, the proof goes through without much adjustments.

3 Translation of Formulas

To translate SGs to proof nets, we first have to translate the formulas of minimal propositional logic into linear logic formulas. This is done via the translation $(-)^*$ defined as follows.

$$A^* := !A \text{ for } A \text{ an atom}$$
$$(A{\rightarrow}B)^* := !(A^{*\perp} \parr B^*)$$

Because we want to unify deduction graph boxes with proof net boxes, other translations seem less suitable.

4 Direct Translation to Proof Nets

In the definition we will use the *rank* of a node: the conclusion nodes of an SG we give rank 0, and – roughly speaking – a node is given rank $i+1$ if there is an arrow to it from a node with rank i. We have to be careful in the case of sharing.

For $\mathcal{B}$ a box in an SG G the *closure of* $\mathcal{B}$, $\overline{\mathcal{B}}$ is the graph consisting of all nodes inside $\mathcal{B}$ plus all nodes that can be reached from within $\mathcal{B}$ in one step. $\overline{\mathcal{B}}$ is also an SG. (This follows from results in [3].)

Definition 10. *Given a deduction graph G, the rank of the nodes in G is defined as follows.*

- *If n is a conclusion node,* $\mathsf{rank}(n) := 0$.
- *If n is a fake conclusion node,* $\mathsf{rank}(n) := 0$.
- *If n is not an* S*-node or an* I*-node and* $\mathsf{rank}(n) = i$ *and* $n \longrightarrow\!\triangleright\, m$, *then* $\mathsf{rank}(m) := i+1$.
- *If n is a* S*-node and* $n \longrightarrow\!\triangleright\, m$, $k \longrightarrow\!\triangleright\, m$, $\mathsf{rank}(n) = i$ *and* $\mathsf{rank}(k) = j$, *then* $\mathsf{rank}(m) := \mathsf{max}(i, j) + 1$.
- *If n is a* I*-node with box $\mathcal{B}$ and* $\mathsf{rank}(n) = i$, *then* $\mathsf{rank}(j) := i+1$ *for all nodes* $j \in \overline{\mathcal{B}}$.

The rank of an SG *G,* $\mathsf{rank}(G)$, *is the maximum of the rank of its nodes.*

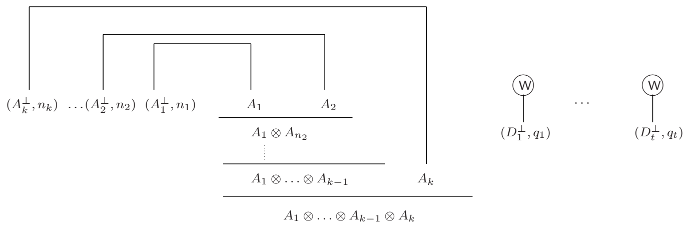
Lemma 11. *Let G be an* SG *with* $\mathsf{rank}(G) = i+1$ *for some i.*

- *Suppose (n, B) of rank i is an* E*-node with edges to (m, A) and $(l, A{\rightarrow}B)$. Then $G \setminus m, l$ is an* SG.
- *Suppose $(n, A{\rightarrow}B)$ of rank i is an* I*-node of box $\mathcal{B}$. Then both $\overline{\mathcal{B}}$ and $G \setminus \overline{\mathcal{B}}$ are* SG*s.*
- *Suppose (m, A) of rank i is an* S*-node with an edge to (n, A). Then $G \setminus n$ is an* SG.
- *Suppose (n, A) of rank i is an* R*-node with an edge to (m, A). Then $G \setminus m$ is an* SG.

Definition 12. *Given an* SG *G we define a proof net (with labelled nodes) $V(G)$, which gives the proof net associated to G, by induction on the number of nodes of G.*
The invariant that we maintain is the following. If G has
– free nodes $(n_1, A_1), \ldots (n_k, A_k)$,
– conclusion nodes $(m_1, B_1), \ldots, (m_l, B_l)$,
then $V(G)$ has
– terminal nodes $B_1^* \otimes \ldots \otimes B_l^*$, and $A_1^{*\perp}, \ldots A_k^{*\perp}$, labelled $n_1, \ldots, n_k$ respectively.

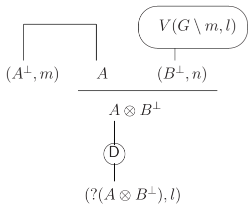
To relieve the notational burden, we just write A instead of A^* in the proof nets. The definition of $V(G)$ is as follows. We make a case-distinction according to $\mathsf{rank}(G)$.

Case $\mathsf{rank}(G) = 0$.
We only have to consider the conclusions $(n_1, A_1), \dots (n_k, A_k)$ of G and the fake conclusions $(q_1, D_1), \dots, (q_t, D_t)$. Now $V(G)$ is the proof net containing axiom links $A_s - A_s^\perp$ (with $A_s^\perp$ labelled by n_s for $1 \le s \le k$) and the $A_1, \dots, A_n$ nodes made into a tensor product $A_1 \otimes \dots \otimes A_n$ by $n-1$ tensor links. Furthermore it contains weakening links on nodes $D_s^\perp$ (labelled n_s) for $1 \le s \le t$.

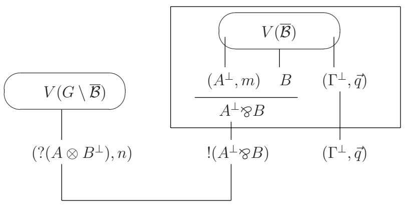


Case $\mathsf{rank}(G) = i + 1$ for some i.
Suppose that $V(F)$ has already been defined for SG*s F with less nodes than G. We consider the non-*A*-nodes of rank i in some order (say the box-topological ordering). We distinguish cases according to the type of the node.*

– E. *Suppose node (n, B) is of rank i with two edges to (m, A) and $(l, A \to B)$. So the nodes m and l are of rank $i+1$. Then:*

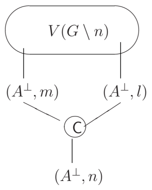


– I. *Suppose node $(n, A \to B)$ of rank i is a box node of box $\mathcal{B}$ with an edge to (j, B) and let (m, A) be the discharged node. Let $(\vec{q}, \Gamma)$ be the nodes that can be reached from within $\mathcal{B}$ with one edge. Then:*

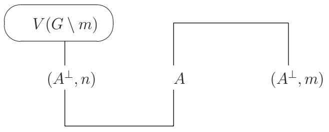


$V(\overline{\mathcal{B}})$
$(A^\perp, m)$
B
$(\Gamma^\perp, \vec{q})$
$A^\perp ⅋ B$
$V(G \setminus \overline{\mathcal{B}})$
$(?(A \otimes B^\perp), n)$
$!(A^\perp ⅋ B)$
$(\Gamma^\perp, \vec{q})$

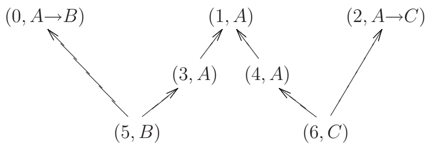
– S *Suppose node (m, A) of rank i is an* S*-node with an edge to (n, A). Let (l, A) be the other* S*-node with the same target. Then:*

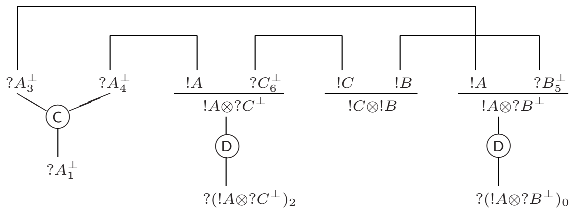


– R *Suppose* (n, A) *of rank* i *is an* R*-node with an edge to* (m, A)*. Then:*



Example 13. Here we see an example of a simple SG G and the translation $V(G)$, written out completely.





5 Translation Via Context Nets

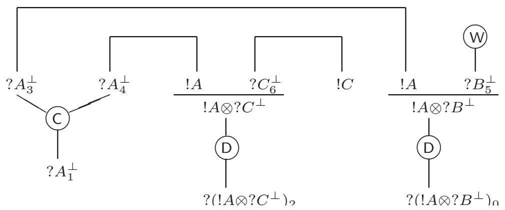
5.1 Context Structures, Context Nets, and Deduction Nets

In [3] we have made a translation that gives the simply typed λ-term associated to *a node* in a deduction graph. This solved the discrepancy that deduction graphs can have many conclusions, whereas simply typed λ-terms have just one type. Another solution could have been to make a translation from deduction graphs to typed λ-terms with conjunction types. So then we would associate a λ-term of type $B_0 \wedge B_1 \wedge \ldots \wedge B_k$ to a deduction graph with conclusions $(n_0, B_0), (n_1, B_1), \ldots, (n_k, B_k)$. In [3] we have not investigated this, but instead we have made a translation from deduction graphs to *contexts*, solving the problem of the many conclusions in a different way. Contexts do not have types (they can only be "well-formed"), but they can obtain a type later, by filling in a variable associated to a node of a deduction graph.

The direct translation of Section 4, can be seen as the proof net equivalent of the translation from DGs to λ-terms with conjunction types. We can also define a translation to proof nets from SGs in a specific conclusion, as shown in

Example 14. This can be seen as the proof net equivalent of the translation to simply typed terms.

Example 14. Here we see a translation of the SG in Example 13 in node 6. The translation of $(5, B)$ is weakened, and thus it plays the role of a fake conclusion.



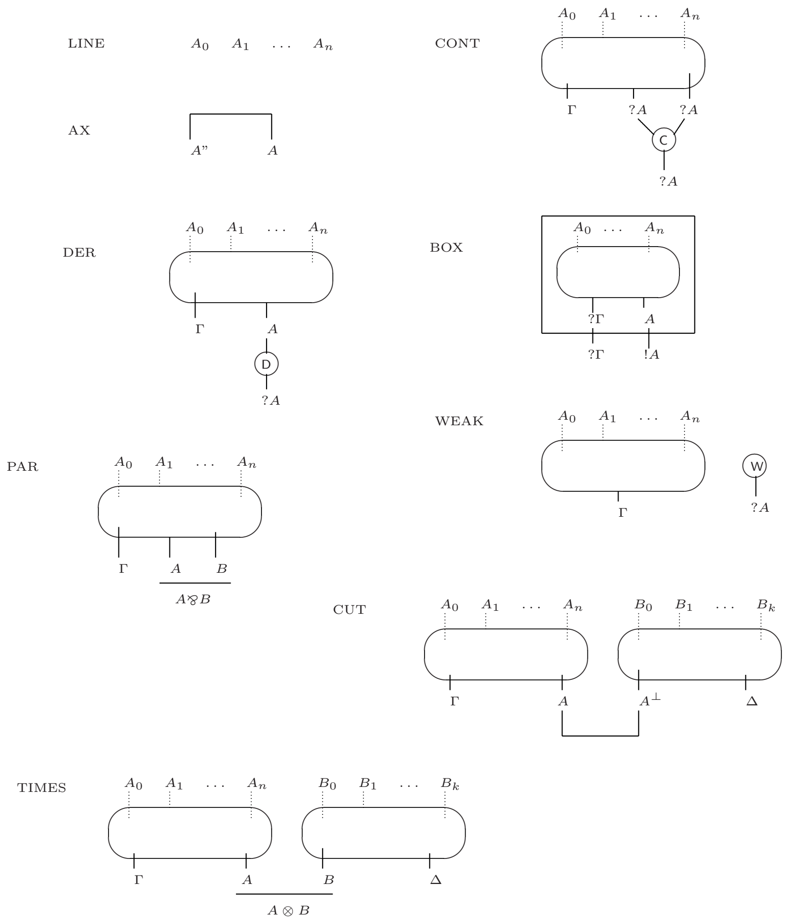
We do not go into this or other translations to proof nets, but we explore a translation to the net equivalent of contexts: context nets. Not only are context nets closer to deduction graphs as they have both "conclusions" and "assumptions", but this translation also generalises the translations to proof nets, because we can recover them by "gluing" something to the context net; the net equivalent of filling a hole (see Section 6).

We define *context structures*, an extension of Girard's proof structures [4]. It allows *initial nodes*, i.e. nodes that are not a conclusion. Similar extensions can be found in Danos and Regnier [1] and Puite [5]. Intuitively, this restores the duality found in two-sided sequents, as initial nodes of a context net can be seen as the formula occurrences left (as long as they are not inside a box) of the turnstyle and the terminal nodes can be seen as the occurrences on the right of it. Another way to look at it, is to consider a context structure as an "open proof structure": if we connect proof structures with corresponding terminal nodes to the initial nodes of the context structure, we get a proof structure. In order to obtain structures that are subgraphs of proof structures, and in contrast to Puite, we do not expand the set of links. Another difference is that we consider an extension of proof nets of MELL (instead of MLL).

Definition 15 (Context Structures). *A* context structure *for MELL is a proof structure for MELL, with the difference that nodes that are not a conclusion, are allowed. These nodes are called* initial nodes*.*

Just as the construction rules of proof nets determine the subset of proof structures that are correct, which means that they agree with a sequent calculus proof, we would like to have construction rules for "context nets", which somehow should determine correct context structures. One could say that context structures are correct, when they agree with a two-sided sequent calculus proof. Another notion of correctness is to say that, if a context net happens to be a proof structure, it is a proof net. It is this latter notion we adopt and we will give the construction rules, which are essentially the rules put forward by David and Kesner [2]. These rules are at the same time a bit rigid, as they consider only the terminal nodes, leaving the initial nodes untouched throughout the construction, and extremely useful, as the class of context structures singled out by these construction rules can easily be seen to be correct.

Definition 16 (Context Nets). Context nets *are inductively defined as follows.*



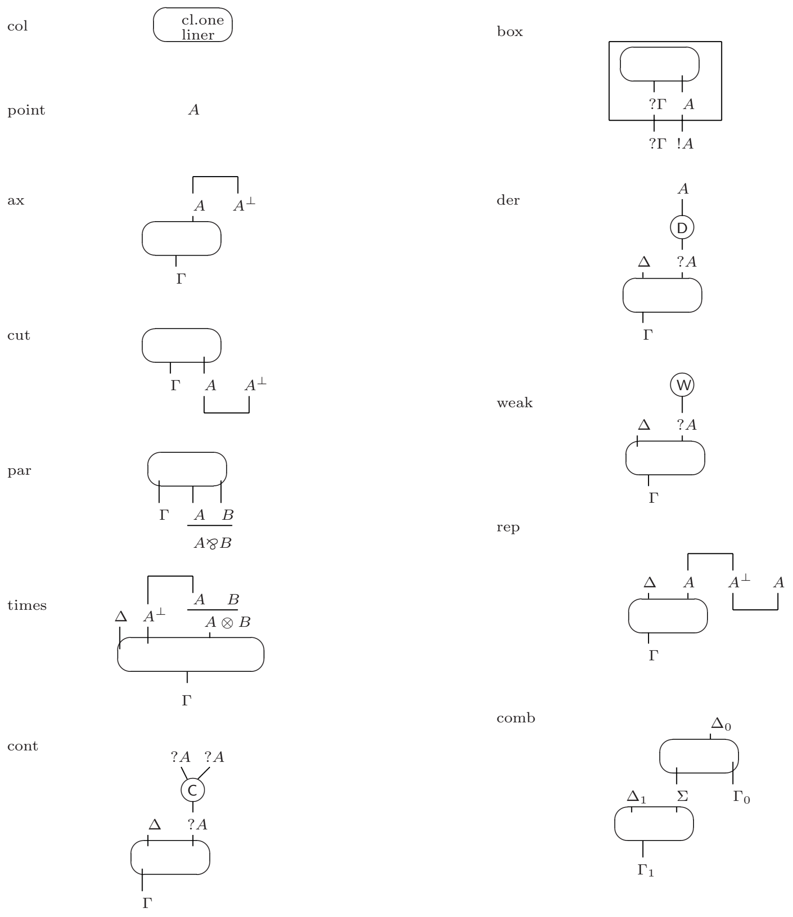
Lemma 17 (Correctness of Context Nets). *Every context net without initial nodes is a proof net.*

Proof. Immediate from the definition of context nets.

We now narrow our attention to a subclass of context nets, the *closed one-liners*. We could see a closed one-liner as a context net Θ such that, if we connect every initial node of it to a corresponding terminal node of *one* proof net, say Σ, we get again a proof net.

Definition 18 (Closed one-liners). *A* closed one-liner *is a context net that has no initial nodes inside boxes and that can be formed by using the LINE-rule at most once.*

Definition 19 (Deduction Nets). Deduction nets *are inductively defined as follows :*



According to the col-rule, every closed one-liner is a deduction net. In the weak-rule, $\Delta \neq \emptyset$. For technical reasons, we only allow the comb-rule when $\Delta_0 = \emptyset$ implies $\Delta_1 = \emptyset$.

Lemma 20. *Every deduction net is a closed one-liner.*

Proof. By induction on the construction of deduction nets. We only treat the interesting cases.

ax By induction, we have a closed one-liner, say C with just one initial node, A. So it uses the LINE-rule with just this A. The new deduction net can be obtained by replacing this LINE-rule in the construction of C by an AX-rule.

cut By induction, we have a closed one-liner, say C, without any initial nodes. So a construction of C does not make use of the LINE-rule. We get the new deduction net by applying LINE on $A^\perp$ and doing a CUT with this on C.

times By induction, we have a closed one-liner, say C, which contains the initial nodes $A^\perp$ and $A \otimes B$. Let Δ be the initial nodes of C without $A^\perp$ and $A \otimes B$. The new deduction net can be obtained by replacing the LINE-rule in the construction of C, by the LINE-rule on Δ, B, an AX-rule and an TIMES-rule.

cont By induction, we have a closed one-liner, say C, with the initial nodes $\Delta, ?A$. Replace the LINE-rule on these nodes by a LINE-rule on $\Delta, ?A, ?A$, followed by a CONT-rule.

der By induction, we have a closed one-liner, say C, with the initial nodes $\Delta, ?A$. Replace the LINE-rule on these nodes by a LINE-rule on Δ, A, followed by a DER-rule.

weak By induction we have a closed one-liner, say C, with initial nodes Δ, $?A$, and terminal nodes Γ. Replace the LINE rule on $\Delta, ?A$, by a LINE-rule on Δ, followed by a WEAK-rule on $?A$.

rep By induction, we have a closed one-liner, say C, with initial nodes Δ, A. Let the LINE-rule of the construction of C be followed by the CUT-rule applied to an A and an $A^\perp$, created by an AX-rule.

comb By induction, we have two closed one-liners, say C_0, with initial nodes Δ_0 and terminal nodes Γ_0, Σ, and C_1, with initial nodes Δ_1, Σ and terminal nodes Γ_1. Then we can replace the LINE-rule of the construction of C_0 with a LINE-rule on Δ_0, Δ_1. Then we get a closed one-liner with terminal nodes $\Gamma_0, \Sigma, \Delta_1$. If we apply the construction of C_1 to this, we obtain the desired result.

Corollary 21. *The class of deduction nets is the class of closed one-liners.*

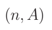
Theorem 22 (Correctness of Deduction Nets). *Every deduction net without initial nodes is a proof net.*

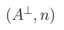
Proof. Immediate from the lemmas 20 and 17.

5.2 From Deduction Graphs to Context Nets

Definition 23. *Given a deduction graph G, we define a deduction net $I(G)$ associated to G by induction on the construction of G. The invariant we maintain is:* **If G is a deduction graph with assumptions Δ and conclusions Γ, then $I(G)$ is a deduction net with terminal nodes $\Delta^{*\perp}$ and initial nodes $\Gamma^{*\perp}$.** *We put a picture of G on the left and a picture of $I(G)$ on the right. Again, A stands for A^* and B stands for B^*.*

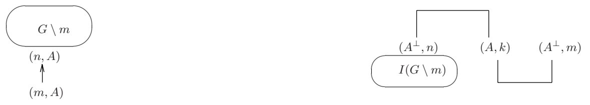
Axiom *The last step is an* **Axiom** *step:*



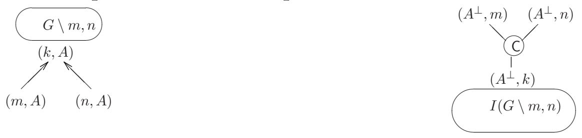


Join *The last step is a* **Join** *step, then by the comb-rule with $\Sigma = \emptyset$:*

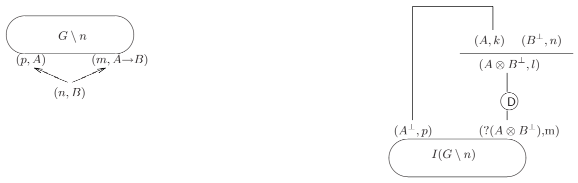
Repeat *The last step is a* **Repeat** *step:*



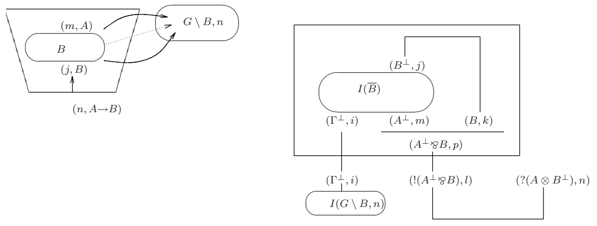
Share *The last step is a* **Share** *step:*



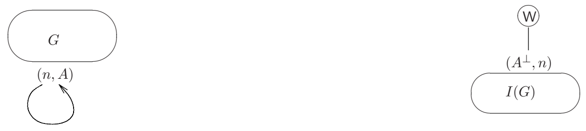
→-E *The last step is a* **→-E** *step:*



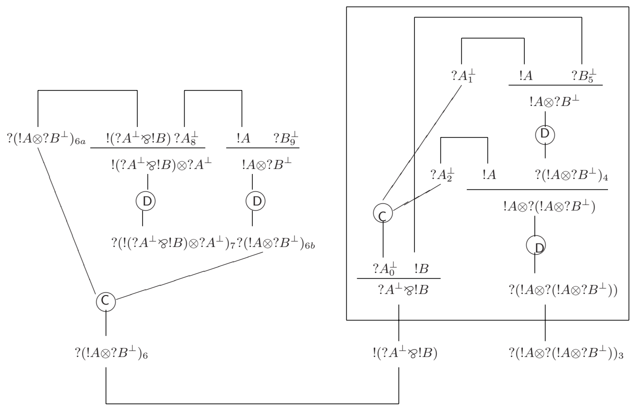
→-I *The last step is a* **→-I** *step:*



Loop *The last step is a* **Loop** *step:*



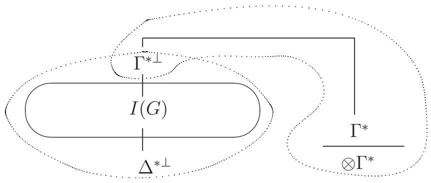
Example 24. We show the proof net that we obtain via $I(-)$ from the example in Figure 1.



$?(!A\otimes ?B^\perp)_{6a}$
$!(?A^\perp ⅋ !B)$
$?A_8^\perp$
$!A$
$?B_9^\perp$
$!(?A^\perp ⅋ !B)\otimes ?A^\perp$
$!A\otimes ?B^\perp$
D
D
$?(!(?A^\perp ⅋ !B)\otimes ?A^\perp)_7$
$?(!A\otimes ?B^\perp)_{6b}$
C
$?(!A\otimes ?B^\perp)_6$
$?A_1^\perp$
$!A$
$?B_5^\perp$
$!A\otimes ?B^\perp$
D
$?A_2^\perp$
$!A$
$?(!A\otimes ?B^\perp)_4$
$!A\otimes ?(!A\otimes ?B^\perp)$
C
D
$?A_0^\perp$
$!B$
$?A^\perp ⅋ !B$
$?(!A\otimes ?(!A\otimes ?B^\perp))$
$!(?A^\perp ⅋ !B)$
$?(!A\otimes ?(!A\otimes ?B^\perp))_3$

6 Connecting the Two Translations

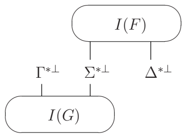
The translations $V(G)$ and $I(G)$ are different in their construction. If G is a deduction graph with assumptions Δ and conclusions Γ, then $I(G)$ is a deduction net with terminal nodes $\Delta^{*\perp}$ and initial nodes $\Gamma^{*\perp}$. By adding axiom links and tensor nodes, we turn this into a proof net with terminal nodes $\Delta^{*\perp}, \bigotimes \Gamma^{*}$, where $\bigotimes \Gamma^{*}$ is the formula obtained by putting a tensor between all formulas in Γ^{*}. This is indicated in the following figure.



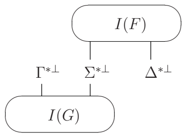
We call this $J(G)$. That this structure is a deduction net follows from the fact that the the two dashed parts are deduction nets and hence the whole structure is, using the comb rule. As it has no initial nodes, it is a proof net. We will see that J and V give the same results.

Lemma 25 (Gluing). *Let G be an* SG *with conclusions Σ, Γ and let F be an* SG *with assumptions Σ, Δ. Define $H := G \cup F$. Then*

1. $I(H) =$



2. $J(H) =$

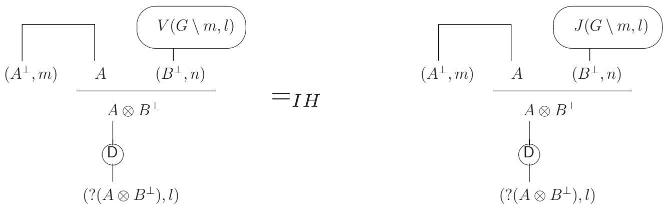


Proof. By induction on the number of non A-nodes of F.

Theorem 26. *Let G be an* SG*. Then $V(G) = J(G)$.*

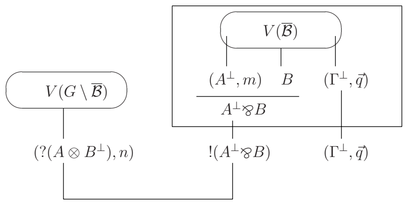
Proof. The proof is by induction on the number of nodes of G. We make a case-distinction to rank(G).
Case rank$(G) = 0$.
Then G has been constructed using **Axiom**, **Join**, and **Loop**. We easily verify that $V(G) = J(G)$.
Case rank$(G) = i + 1$ for some i.
Suppose that $V(F)$ has already been defined for SGs F with less nodes than G. We consider the non-A-nodes in some order (say the box-topological ordering). We distinguish cases according to the type of the node. We treat here the E-case and the I-case.

– E Suppose node (n, B) is of rank i with an edge to (m, A) and an edge to $(l, A{\to}B)$. Let F be the graph consisting of the nodes n, m, and l and the edges $n \longrightarrow\!\triangleright\, m$ and $n \longrightarrow\!\triangleright\, l$. Then $V(G) =$



Note that this is $J(G \setminus m, l)$ glued to $I(F)$ and by Lemma 25 we are done.

– I Suppose node $(n, A{\to}B)$ of rank i is a box-node of box $\mathcal{B}$ with an edge to (j, B) and let (m, A) be the discharged node. Let $(\vec{q}, \Gamma)$ be the nodes that can be reached from $\mathcal{B}$ with one edge.


$V(\overline{\mathcal{B}})$
$V(G \setminus \overline{\mathcal{B}})$
$(A^\perp, m)$
B
$(\Gamma^\perp, \vec{q})$
$A^\perp ⅋ B$
$(?(A \otimes B^\perp), n)$
$!(A^\perp ⅋ B)$
$(\Gamma^\perp, \vec{q})$

Let F be the graph consisting of box $\mathcal{B}$ with box-node n and all nodes that are reachable from $\mathcal{B}$ within one step. By induction we conclude that $V(G)$ is $J(G \setminus \overline{\mathcal{B}})$ glued to $I(F)$. By Lemma 25 we are done.

7 Preservation of Reduction

We will show that the transformations involved in the process of cut-elimination of SGs can be mimicked by reductions on their context net translations. Of course this implies that it can be mimicked by reductions on the result of the direct translation of Section 4 too, and indeed on any translation that is an extension of the translation via context nets.

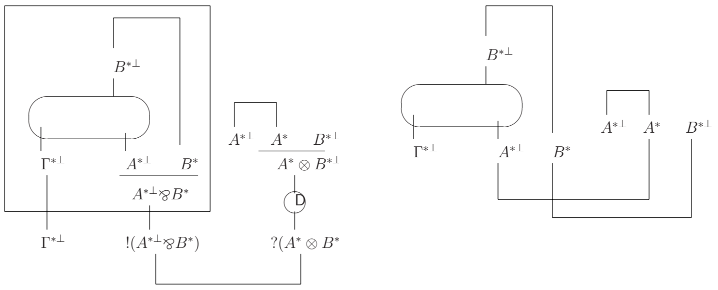
The notion of reduction on proof structures can be extended to reduction on context structures without any problem. We will use the following reduction rules: Ax-cut, ⅋-⊗, d-b, c-b and b-b. (See [4] and [3].)

Theorem 27. *The class of deduction nets is closed under reduction.*

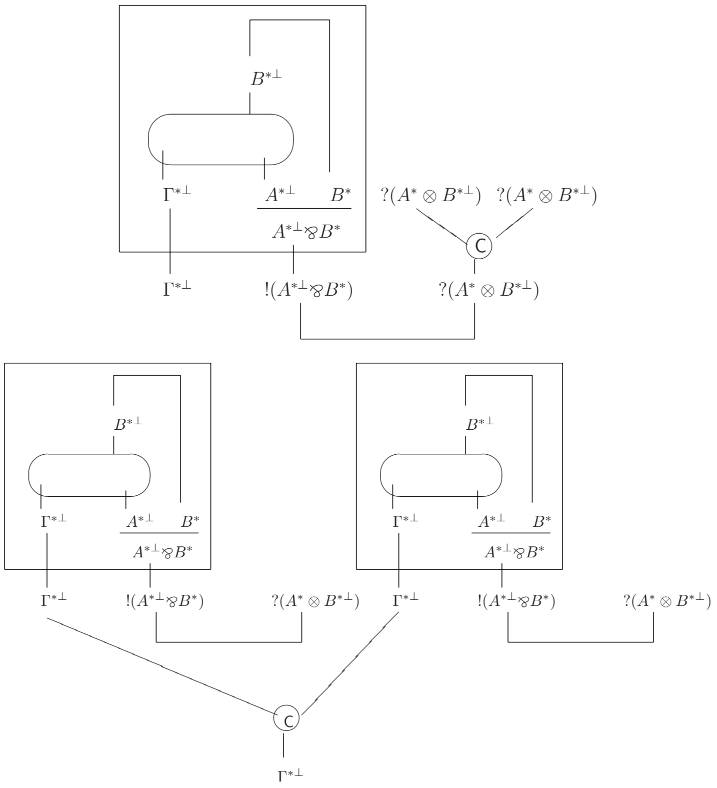
Proof. Note that we can make a proof net from any given deduction net by the comb-rule. Because all reduction rules preserve initial and terminal nodes, it now follows that the class of deduction nets is closed under reduction.

Theorem 28. *If G' is obtained from G by either elimination of a safe cut, an unsharing step, a repeat-elimination step, or an incorporation step, then $I(G')$ can be obtained from $I(G)$ by one or more reductions of context nets.*

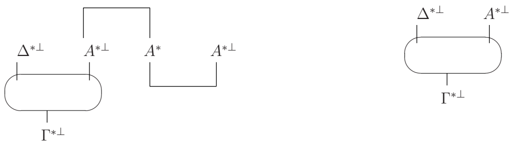
Proof. – If G' is obtained from G by eliminating a safe cut, then there exists a deduction net F such that $I(G) \rightarrow\!\!\!>_{d-b} F \rightarrow\!\!\!>_{\wp-\otimes} I(G'')$.



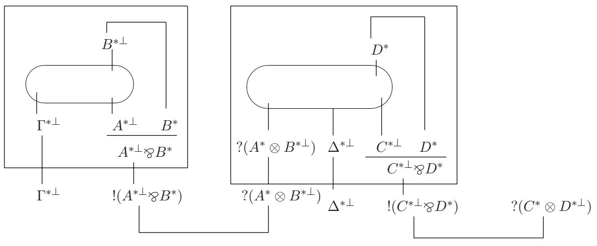
– If G' is obtained from G by unsharing, then $I(G) \rightarrow\!\!\!>_{c-b} I(G')$.

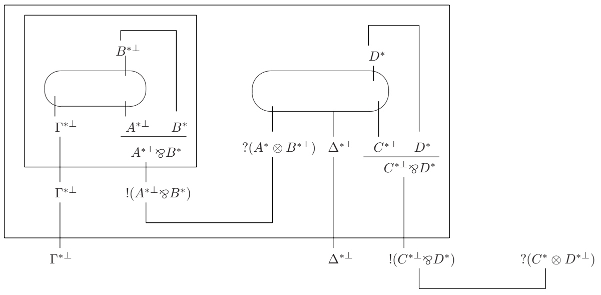

$B^{*\perp}$
$\Gamma^{*\perp}$
$A^{*\perp}$ B^*
$A^{*\perp}\wp B^*$
$?(A^* \otimes B^{*\perp})$ $?(A^* \otimes B^{*\perp})$
C
$\Gamma^{*\perp}$ $!(A^{*\perp}\wp B^*)$ $?(A^* \otimes B^{*\perp})$
$B^{*\perp}$
$\Gamma^{*\perp}$ $A^{*\perp}$ B^*
$A^{*\perp}\wp B^*$
$\Gamma^{*\perp}$ $!(A^{*\perp}\wp B^*)$ $?(A^* \otimes B^{*\perp})$
$B^{*\perp}$
$\Gamma^{*\perp}$ $A^{*\perp}$ B^*
$A^{*\perp}\wp B^*$
$\Gamma^{*\perp}$ $!(A^{*\perp}\wp B^*)$ $?(A^* \otimes B^{*\perp})$
C
$\Gamma^{*\perp}$

– If G' is obtained from G by a repeat elimination, then $I(G) \rightarrow\!\!\!>_{Ax-cut} I(G')$.


$\Delta^{*\perp}$ $A^{*\perp}$ A^* $A^{*\perp}$
$\Gamma^{*\perp}$
$\Delta^{*\perp}$ $A^{*\perp}$
$\Gamma^{*\perp}$

– If G' is obtained from G by incorporation, then $I(G) \twoheadrightarrow_{b-b} I(G')$.





From this correspondence of reduction follows again strong normalisation for the process of cut-elimination of SGs. However, the result is stronger than the one mentioned in Section 2. There we assumed the total removal of a cut before starting the process of cut-elimination for another cut. Moreover, the process of cut-elimination prescribes an order of the various steps. Seeing these steps of the process as separate reduction steps, one could say that the process describes a reduction *strategy* and the normalisation result is rather weak normalisation.

But now we see that the steps may be done in any order, even mixing the steps for various cuts. This gives us strong normalisation in a very general sense.

Acknowledgments

We thank Delia Kesner, Stéphane Lengrand, and Femke van Raamsdonk for their valuable suggestions.

References

1. V.Danos and L.Regnier, *The structure of the multiplicatives*, Archive for Mathematical logic 28, 1989.
2. R.David and D.Kesner, *An arithmetical strong-normalisation proof for reduction modulo in proof-nets*, draft.
3. H.Geuvers and I.Loeb, *Natural Deduction via Graphs: Formal Definition and Computation Rules*, to appear in MSCS (http://www.cs.ru.nl/~herman/PUBS/gd.pdf)
4. J.-Y. Girard, Linear Logic, *Theoretical Computer Science*, 50(1):1-101, 1987.
5. Q.Puite, *Proof Nets with Explicit Negation for Multiplicative Linear Logic*, Preprint 1079, Department of Mathematics, Utrecht University, 1998.

The Structure of Tractable Constraint Satisfaction Problems

Martin Grohe

Institut für Informatik, Humboldt Universität
Unter den Linden 6, 10099 Berlin, Germany

Abstract. We give a survey of recent results on the complexity of constraint satisfaction problems. Our main emphasis is on tractable structural restrictions.

1 Introduction

The objective of a *constraint satisfaction problem (CSP)* is to assign values to variables subject to constraints on the values. Obviously, this is a very general type of problem, and it is not surprising that many algorithmic problems in various areas of computer science can be described as CSPs. It is neither surprising that, in general, CSPs are computationally hard. Considerable efforts have been made to precisely understand the complexity of CSPs, with the goal of identifying tractable restrictions (often referred to as "islands of tractability" in this context) and, ultimately, determining the boundary between tractable and intractable CSPs. There are two main types of restrictions that have been studied: *Constraint language restrictions*, which are concerned with the types of constraints that occur, and *structural restrictions*, which are concerned with the structure induced by the constraints on the variables, for example, with the way the constraints overlap.

1.1 CSP-Instances

An *instance* of a CSP is a triple (V, D, C) consisting of a set V of *variables*, a *domain* D, and a set C of *constraints*. The objective is to find an assignment of values from D to the variables such that all constraints in C are satisfied. The constraints are expressions of the form $Rx_1 \dots x_k$, where R is a k-ary relation on D and $x_1, \dots, x_k$ are variables. A constraint $Rx_1 \dots x_k$ is satisfied if the k-tuple of values assigned to the variables $x_1, \dots, x_k$ belongs to the relation R. The *constraint language* of a CSP-instance (V, D, C) is the set of all relations that occur in the constraints in C.

Example 1. We describe Sat, the satisfiability problem for Boolean formulas in conjunctive normal form (CNF), as a CSP: A CNF-formula ϕ corresponds to a CSP-instance whose variables are the variables of ϕ, whose domain is $\{0, 1\}$, and whose constraints are given by the clauses. For example, the clause $(x \vee \neg y \vee \neg z)$ corresponds to a constraint $Rxyz$, where R is the ternary relation $\{(0,0,0), (0,0,1), (0,1,0), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$ on $\{0, 1\}$.

Example 2. As a second example, we consider the 3-Colourability problem for graphs. Its objective is to colour the vertices of a graph with 3 colours in such a way that adjacent vertices get different colours. Each graph $\mathcal{G}$, viewed as an instance of 3-Colourability, corresponds to the following CSP-instance: Variables are the vertices of $\mathcal{G}$, the domain is the set of colours, say, $\{\text{red, blue, green}\}$, and for each edge $\{v, w\}$ of $\mathcal{G}$ there is a constraint Ivw, where I is the disequality relation on the domain.

Example 3. As a third example, consider the Clique problem, whose objective it is to decide whether a graph $\mathcal{G}$ has a clique of size k. An instance $(\mathcal{G}, k)$ of Clique corresponds to the following CSP-instance: The variables are $x_1, \dots, x_k$, the domain is the vertex set of $\mathcal{G}$, and there are constraints Ex_ix_j for $1 \leq i \leq j$, where E is the edge relation of $\mathcal{G}$.

1.2 Relational Structures and Homomorphisms

It is a well known observation (going back to Feder and Vardi [16]) that CSPs can be described as homomorphism problems for relational structures (and vice versa). A *(relational) vocabulary* is a finite set τ of relation symbols, each with a prescribed arity. A *structure* $\mathcal{A}$ of vocabulary τ (for short: τ-structure) consists of a finite set A, called the *universe* of $\mathcal{A}$, and a k-ary relation $r^{\mathcal{A}}$ for each k-ary relation symbol $r \in \tau$. A *homomorphism* from a τ-structure $\mathcal{A}$ to a τ-structure $\mathcal{B}$ is a mapping $h : A \to B$ from the universe of $\mathcal{A}$ to the universe of $\mathcal{B}$ that preserves all relations, that is, for all $r \in \tau$, say, of arity k, and all tuples $(a_1, \dots, a_k) \in r^{\mathcal{A}}$ it holds that $(h(a_1), \dots, h(a_k)) \in r^{\mathcal{B}}$.

With every CSP-instance $\mathcal{I} = (V, D, \mathcal{C})$ we associate two structures $\mathcal{A}(\mathcal{I})$ and $\mathcal{B}(\mathcal{I})$ as follows: The vocabulary $\tau(\mathcal{I})$ of both structures contains a k-ary relation symbol r for every k-ary relation R in the constraint language of $\mathcal{I}$. The universe of $\mathcal{B}(\mathcal{I})$ is D, and the relations of $\mathcal{B}$ are those appearing in the constraint language. More precisely, for every $r \in \tau(\mathcal{I})$ we let $r^{\mathcal{B}(\mathcal{I})} = R$. The universe of $\mathcal{A}(\mathcal{I})$ is V, and for each k-ary relation symbol $r \in \tau(\mathcal{I})$ we let $r^{\mathcal{A}(\mathcal{I})} = \{(x_1, \dots, x_k) \mid Rx_1 \dots x_k \in \mathcal{C}\}$. Then a mapping $h : V \to D$ is a satisfying assignment for the CSP-instance $\mathcal{I}$ if and only if it is a homomorphism from $\mathcal{A}(\mathcal{I})$ to $\mathcal{B}(\mathcal{I})$. Thus $\mathcal{I}$ is satisfiable if and only if there is a homomorphism from $\mathcal{A}(\mathcal{I})$ to $\mathcal{B}(\mathcal{I})$. Conversely, for all pairs of structures $\mathcal{A}$, $\mathcal{B}$ of the same vocabulary, we can easily construct a CSP-instance $\mathcal{I}$ such that $\mathcal{A}(\mathcal{I}) = \mathcal{A}$ and $\mathcal{B}(\mathcal{I}) = \mathcal{B}$.

1.3 Restricted CSPs

The structures $\mathcal{A}(\mathcal{I})$ and $\mathcal{B}(\mathcal{I})$ associated with a CSP-instance $\mathcal{I}$ precisely reflect the two kinds of restrictions on CSPs studied in the literature on the complexity of CSPs: $\mathcal{B}(\mathcal{I})$ is just a fancy representation of the constraint language. Hence restrictions on $\mathcal{B}(\mathcal{I})$ are *constraint language restrictions.* $\mathcal{A}(\mathcal{I})$ is the "structure induced by the constraints on the variables", made precise. Hence restrictions on $\mathcal{A}(\mathcal{I})$ are *structural restrictions.*

In general, for all classes C and D of structures, we have the following restricted CSP:

$\textsc{Csp}(\mathsf{C}, \mathsf{D})$
Instance: CSP-instance $\mathcal{I}$ with $\mathcal{A}(\mathcal{I}) \in \mathsf{C}$ and $\mathcal{B}(\mathcal{I}) \in \mathsf{D}$.
Problem: Decide if $\mathcal{I}$ is satisfiable.

We write $\textsc{Csp}(-, \mathsf{D})$ instead of $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ if C is the class of all structures and $\textsc{Csp}(\mathsf{C}, -)$ if D is the class of all structures. For structures $\mathcal{B}$, we write $\textsc{Csp}(\mathsf{C}, \mathcal{B})$ and $\textsc{Csp}(-, \mathcal{B})$ instead of $\textsc{Csp}(\mathsf{C}, \{\mathcal{B}\})$ and $\textsc{Csp}(-, \{\mathcal{B}\})$. Restrictions of the form $\textsc{Csp}(\mathsf{C}, \mathcal{B})$ are called *nonuniform* [27].

Example 2 revisited: The 3-Colourability problem corresponds to $\textsc{Csp}(\mathsf{G}, \mathcal{D})$, where G denotes the class of all (undirected, loop-free) graphs and $\mathcal{D}$ is a triangle. To see this, note that the disequality relation on a three element domain corresponds to a triangle if viewed as a graph. Since only graphs can be mapped homomorphically to a triangle, $\textsc{Csp}(\mathsf{G}, \mathcal{D})$ is essentially the same problem as $\textsc{Csp}(-, \mathcal{D})$. Thus 3-Colourability is a nonuniform CSP with a restricted constraint language.

Example 3 revisited: The Clique problem corresponds to $\textsc{Csp}(\mathsf{K}, \mathsf{G})$, where K denotes the class of all complete graphs. This problem is essentially the same problem as $\textsc{Csp}(\mathsf{K}, -)$. Thus Clique is a CSP with a restricted structure.

Example 1 revisited: The description of Sat as a problem $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ is slightly more complicated: The *sign pattern* of a clause $\gamma = (\lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_n)$ is the tuple $\sigma(\gamma) = (s_1, \ldots, s_n)$ of signs, where $s_i = +$ if λ_i is a positive literal (that is, a variable) and $s_i = -$ if λ_i is a negative literal (that is, a negated variable). For example, the sign pattern of the clause $(x \vee \neg y \vee \neg z)$ is $(+, -, -)$. With each n-ary sign pattern σ we associate an n-ary relation R_σ over the Boolean domain that consists of all satisfying assignments for clauses with sign pattern σ. For example, $R_{(+,-,-)} = \{0,1\}^3 \setminus \{(0,1,1)\}$.

Now let $\mathcal{I}$ be the CSP-instance associated with a CNF-formula ϕ. The vocabulary $\tau(\mathcal{I})$ consists of an n-ary relation symbol r_σ for every sign pattern σ that occurs in ϕ. Furthermore, $\mathcal{B}(\mathcal{I})$ is the $\tau(\mathcal{I})$-structure with universe $\{0,1\}$ and relations $r_\sigma^{\mathcal{B}(\mathcal{I})} = R_\sigma$. Let S be the class of all structures $\mathcal{B}$ whose universe is $\{0,1\}$ and whose vocabulary consists of finitely many relation symbols r_σ for sign patterns σ, such that $r_\sigma^{\mathcal{B}} = R_\sigma$. Then Sat corresponds to $\textsc{Csp}(-, \mathsf{S})$.[1]

The examples show that problems $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ are NP-hard in general. Furthermore, both structural restrictions $\textsc{Csp}(\mathsf{C}, -)$, such as Clique, and constraint language restrictions $\textsc{Csp}(-, \mathsf{D})$, such as Sat, can be NP-hard. Even nonuniform constraint language restrictions $\textsc{Csp}(-, \mathcal{B})$, such as 3-Colourability, can be NP-hard. However, observe that "nonuniform structural restrictions" of the form $\textsc{Csp}(\{\mathcal{A}\}, -)$ are always in PTIME, because the set of variables is fixed.

The reader may wonder why we write "NP-hard" instead of "NP-complete" — it seems obvious that all problems $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ are in NP. However, this is not entirely true, because the membership problem for the classes C and D may not be in NP (it may even be undecidable), and in this case it is not even decidable

[1] The astute reader may notice that the translation from Sat to $\textsc{Csp}(-, \mathsf{S})$ involves an exponential blow-up in size. We shall discuss this issue in Sec. 3.1.

in NP if a given CSP-instance $\mathcal{I}$ is an instance of $\textsc{Csp}(\mathsf{C}, \mathsf{D})$. We could avoid this problem by requiring the classes C and D to be polynomial time decidable, but there are interesting examples where they are not. Instead, we view $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ as a promise problem: We say that $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ is *solvable in polynomial time* if there is a polynomial time algorithm that, given an instance $\mathcal{I}$ with $\mathcal{A}(\mathcal{I}) \in \mathsf{C}$ and $\mathcal{B}(\mathcal{I}) \in \mathsf{D}$, correctly decides if $\mathcal{I}$ is solvable. We do not care what the algorithm does if the input is not of this form.

Definition 4. Let C, D be classes of structures. Then $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ is *tractable* if it is solvable in polynomial time (viewed as a promise problem) and *intractable* otherwise. $\textsc{Csp}(\mathsf{C}, \mathsf{D})$ is *hard* if it is NP-hard.

2 Constraint Language Restrictions

Most and the mathematically deepest work on the complexity of CSPs is concerned with constraint language restrictions (e.g., [7,8,6,4,13,16,22,26,29]). As a matter of fact, most of this work is only concerned with nonuniform constraint language restrictions of the form $\textsc{Csp}(-, \mathcal{B})$. The driving force behind this work is a conjecture that has first been stated by Feder and Vardi in 1993 (in the conference version of [16]):

Conjecture 1 (Dichotomy Conjecture). *For every structure $\mathcal{B}$, the problem $\textsc{Csp}(-, \mathcal{B})$ is either tractable or hard.*

In other words: Every problem of the form $\textsc{Csp}(-, \mathcal{B})$ is either in PTIME or NP-complete (as all such problems are contained in NP). A priori, there is no reason why this should be true, in particular in view of Ladner's theorem [28] stating that if PTIME $\neq$ NP, then there are problems in NP that are neither in PTIME nor NP-complete. Nevertheless, the Dichotomy Conjecture still stands unrefuted, and it has actually been proved in several significant special cases.

Two important special cases of the conjecture had already been proved when Feder and Vardi stated it. In 1978, Schaefer [29] studied what he called "generalised satisfiability problems". In our terminology, these are just CSPs over the Boolean domain $\{0, 1\}$. Let us call a structure $\mathcal{B}$ *Boolean* if its universe is $\{0, 1\}$.

Theorem 5 (Schaefer [29]). *The dichotomy conjecture holds for all Boolean structures. More precisely, for every Boolean structure $\mathcal{B}$, the problem $\textsc{Csp}(-, \mathcal{B})$ is tractable if $\mathcal{B}$ satisfies one of the following conditions:*

(1) Every relation of $\mathcal{B}$ contains a tuple in which all entries are 0.
(2) Every relation of $\mathcal{B}$ contains a tuple in which all entries are 1.
(3) Every relation of $\mathcal{B}$ is definable by a CNF-formula in which every clause contains at most one negative literal.
(4) Every relation of $\mathcal{B}$ is definable by a CNF-formula in which every clause contains at most one positive literal.
(5) Every relation of $\mathcal{B}$ is definable by a CNF-formula in which every clause contains at most two literals.

(6) Every relation of $\mathcal{B}$ is the set of solutions of a system of linear equations over the two element field GF(2)*.*

Otherwise, $\textsc{Csp}(-, \mathcal{B})$ *is hard.*

Obviously, Schaefer's theorem implies the Dichotomy conjecture for all two-element structures $\mathcal{B}$. It is not hard to see that Schaefer's theorem also implies a dichotomy for all classes of Boolean structures: For every class B of Boolean structures, $\textsc{Csp}(-, \mathsf{B})$ is tractable if every structure $\mathcal{B} \in \mathsf{B}$ satisfies one of the conditions (1)–(6) and hard otherwise.

Note that $\textsc{Csp}(-, \mathcal{B})$ is tractable for all one-element structures $\mathcal{B}$. Hence, for the rest of this section we assume that all structures have at least two elements.

Hell and Nešetřil [22] studied the complexity of the graph homomorphism problem for a fixed target graph $\mathcal{H}$ (the so called *$\mathcal{H}$-colouring problem*) and proved a dichotomy result. In our terminology, their result reads as follows:

Theorem 6 (Hell and Nešetřil [22]). *The dichotomy conjecture holds for graphs. More precisely, for every graph $\mathcal{H}$, the problem* $\textsc{Csp}(-, \mathcal{H})$ *is tractable if $\mathcal{H}$ is bipartite; otherwise, it is hard.*

Remember that we assume graphs to be undirected an loop-free. The dichotomy can easily be extend to graphs with loops, because $\textsc{Csp}(-, \mathcal{H})$ is tractable for every $\mathcal{H}$ that has a loop. Hell and Nešetřil's theorem also implies a dichotomy for classes of graphs: For every class H of graphs, $\textsc{Csp}(-, \mathsf{H})$ is tractable if every $\mathcal{H} \in \mathsf{H}$ is bipartite and hard otherwise. For directed graphs, the situation is much more complicated: Feder and Vardi [16] proved that the dichotomy conjecture for directed acyclic graphs is equivalent to the general dichotomy conjecture. Even for oriented trees, the conjecture is still open (cf. [23,24]).

In recent years, Bulatov, Cohen, Dalmau, Jeavons, Krokhin and others very successfully pursued an algebraic approach to the complexity of CSPs. While still short of proving the dichotomy conjecture, they made some remarkable progress. We cannot hope to explain the approach in any depth in this short survey. For detailed presentations of the algebraic approach, we refer the reader to [8,11,26]. Our modest goal for the rest of this section is to state the main results obtained by this approach and the currently conjectured dividing line between tractable and hard constraint language restrictions.

The *ℓth power* of a τ-structure $\mathcal{B}$ is the τ-structure $\mathcal{B}^\ell$ with universe B^ℓ and relations

$$r^{\mathcal{B}^\ell} = \Big\{ \big((b_{11}, \dots, b_{1\ell}), \dots, (b_{k1}, \dots, b_{k\ell})\big) \;\Big|\; (b_{1j}, \dots, b_{kj}) \in r^{\mathcal{B}} \text{ for } 1 \le j \le \ell \Big\}$$

for every k-ary $r \in \tau$. An *(ℓ-ary) polymorphism* of $\mathcal{B}$ is a homomorphism from $\mathcal{B}^\ell$ to $\mathcal{B}$. Equivalently, an ℓ-ary polymorphism of $\mathcal{B}$ can be described as an ℓ-ary operation h on B (that is, a mapping $h : B^\ell \to B$) such that for every k, every k-ary $r \in \tau$, and every matrix $(b_{ij})_{\substack{1 \le i \le k \\ 1 \le j \le \ell}} \in B^{k \times \ell}$ the following holds: If all columns of the matrix are in $r^{\mathcal{A}}$, then the k-tuple obtained by applying h to each row of the matrix is in $r^{\mathcal{A}}$.

Example 7. Consider the Boolean structure $\mathcal{B}$ with one ternary relation

$$R = \big\{(0,1,0),(0,1,0),(1,0,0),(1,1,1)\big\}$$

Observe that R is the set of solutions to the linear equation $x_1 + x_2 + x_3 = 1$ over GF(2). Then the mapping $f : \{0,1\}^2 \to \{0,1\}$ defined by $f(x,y) = x+y+1$ (addition in GF(2)) is a binary polymorphism of $\mathcal{B}$. To see this, note that if the two columns of the matrix

$$\begin{pmatrix} b_1 & c_1 \\ b_2 & c_2 \\ b_3 & c_3 \end{pmatrix}$$

solve the equation $x_1 + x_2 + x_3 = 1$, then so does the vector

$$\begin{pmatrix} f(b_1,c_1) \\ f(b_2,c_2) \\ f(b_3,c_3) \end{pmatrix} = \begin{pmatrix} b_1 + c_1 + 1 \\ b_2 + c_2 + 1 \\ b_3 + c_3 + 1 \end{pmatrix}$$

The set of all polymorphisms of a structure $\mathcal{B}$, denoted by $\mathrm{Pol}(\mathcal{B})$, is called the *clone* of $\mathcal{B}$. The following lemma establishes a fundamental connection between the complexity of $\textsc{Csp}(-,\mathcal{B})$ and the clone of $\mathcal{B}$ that underlies the algebraic approach:

Lemma 8 (Jeavons [25]). *Let $\mathcal{B},\mathcal{B}'$ be structures over the same universe. If $\mathrm{Pol}(\mathcal{B}) \subseteq \mathrm{Pol}(\mathcal{B}')$ then $\textsc{Csp}(-,\mathcal{B}')$ is polynomial time reducible to $\textsc{Csp}(-,\mathcal{B})$.*

In particular, if $\mathrm{Pol}(\mathcal{B}) = \mathrm{Pol}(\mathcal{B}')$ then $\textsc{Csp}(-,\mathcal{B})$ and $\textsc{Csp}(-,\mathcal{B}')$ are polynomial time equivalent.

In other words: The larger the clone of a structure, the simpler the corresponding CSP. Intuitively, this is plausible, because more complex relations will have fewer polymorphisms. Note that the clone of every structure contains all projections $f(x_1,\ldots,x_\ell) = x_i$. Thus if the clone of a structure $\mathcal{B}$ (with at least two elements) contains only the projections, then $\textsc{Csp}(-,\mathcal{B})$ is certainly hard. The converse of this observation fails; it is easy to construct structures $\mathcal{B}$ such that $\textsc{Csp}(-,\mathcal{B})$ is hard and $\mathrm{Pol}(\mathcal{B})$ does not only contain projections.

An *algebra* is a pair (A,F) consisting of a set A and a set F of operations on A. With every structure $\mathcal{B}$ we associate the algebra $\mathbf{A}(\mathcal{B}) = (B,\mathrm{Pol}(\mathcal{B}))$. It follows immediately from Lemma 8 that for all structures $\mathcal{B},\mathcal{B}'$, if $\mathbf{A}(\mathcal{B}) = \mathbf{A}(\mathcal{B}')$ then $\textsc{Csp}(-,\mathcal{B})$ and $\textsc{Csp}(-,\mathcal{B}')$ are polynomial time equivalent.

The set of *term operations* of an algebra (A,F) is the closure of F and all projections under composition. Since a clone contains all projections and is closed under composition, the term operations of $\mathbf{A}(\mathcal{B})$ are precisely the operations in $\mathrm{Pol}(\mathcal{B})$. An algebra is *idempotent* if all its term operations are idempotent, that is, $f(a,\ldots,a) = a$ for all a.

Lemma 9 (Bulatov, Jeavons, and Krokhin [8]). *For every structure $\mathcal{B}$ there exists a structure $\mathcal{B}'$ such that $\mathbf{A}(\mathcal{B}')$ is idempotent, and $\textsc{Csp}(-,\mathcal{B})$ and $\textsc{Csp}(-,\mathcal{B}')$ are polynomial time equivalent.*

For the readers familiar with the terminology, let us remark that the structure $\mathcal{B}'$ of Lemma 9 is an expansion of the *core* of $\mathcal{B}$ by unary relations $\{b\}$ for all elements b. In particular, this implies that the cardinality of the universe of $\mathcal{B}'$ is less than or equal to the cardinality of the universe of $\mathcal{B}$.

A *subalgebra* of an algebra (A, F) is an algebra (A', F'), where A' is a subset of A that is closed under all operations in F and F' consists of the restrictions of the operations in F to A'. A *homomorphic image* of (A, F) is an algebra (A', F') such that there exist surjective mappings $h : A \to A'$ and $\iota : F \to F'$ such that for all ℓ-ary operations $f \in F$, $\iota(f)$ is an ℓ-ary operation on A' with

$$h(f(a_1, \ldots, a_\ell)) = \iota(f)(h(a_1), \ldots, h(a_\ell))$$

for all tuples $(a_1, \ldots, a_\ell) \in A^\ell$. A *factor* of an algebra (A, F) is a homomorphic image of a subalgebra of (A, F). A factor is *nontrivial* if it has at least two elements. Now we are ready to state the main conjecture by Bulatov, Jeavons, and Krokhin [8]:

Conjecture 2 (BJK-Conjecture). *For every structure $\mathcal{B}$, if $\mathbf{A}(\mathcal{B})$ is idempotent, then the problem* $\textsc{Csp}(-, \mathcal{B})$ *is hard if $\mathbf{A}(\mathcal{B})$ has a nontrivial factor all of whose operations are projections, and tractable otherwise.*

By Lemma 9, the BJK-Conjecture implies the Dichotomy Conjecture. But in addition, the BJK-Conjecture predicts where the dividing line between tractable and hard instances is located. Bulatov, Jeavons, and Krokhin [8] proved one direction of the BJK-Conjecture: If $\mathbf{A}(\mathcal{B})$ has a nontrivial factor all of whose operations are projections, then $\textsc{Csp}(-, \mathcal{B})$ is hard. It follows from Theorems 5 that the BJK-Conjecture holds for all two-element structures.

Theorem 10 (Bulatov [6]). *The BJK-Conjecture (and hence the Dichotomy Conjecture) holds for all three-element structures $\mathcal{B}$.*

Bulatov also proved the BJK-Conjecture for graphs [5] and so-called *conservative CSPs* [4], in which the set of values for each variable can be restricted arbitrarily. Conservative CSPs can be characterised as problems $\textsc{Csp}(-, \mathcal{B})$ for structures $\mathcal{B}$ such that every ℓ-ary polymorphism f satisfies $f(b_1, \ldots, b_\ell) \in \{b_1, \ldots, b_\ell\}$ for all $b_1, \ldots, b_\ell \in B$.

3 Structural Restrictions

We start with an example that illustrates a simple, but important algorithmic idea:

Example 11. A *labelled tree* is a structure $\mathcal{T} = (T, E, P_1, \ldots, P_n)$, where E is binary, $n \geq 0$, and $P_1, \ldots, P_n$ are unary, such that the restriction (T, E) is a tree. Let T denote the class of all labelled trees. Then $\textsc{Csp}(\mathsf{T}, -)$ is tractable.

To see this, it will be most convenient to view $\textsc{Csp}(\mathsf{T}, -)$ as a homomorphism problem (as described in Section 1.2). The input consists of a labelled tree $\mathcal{T} =$

$(T, E, P_1, \dots, P_n)$ and a structure $\mathcal{B} = (B, F, Q_1, \dots, Q_n)$, where without loss of generality we assume that $\mathcal{T}$ and $\mathcal{B}$ have the same vocabulary. We fix an arbitrary root r for $\mathcal{T}$ and direct the edges away from the root. For $t \in T$, let $\mathcal{T}_t$ denote the induced subtree of $\mathcal{T}$ whose nodes are t and all its descendants in $\mathcal{T}$. For $t \in T$, $b \in B$, we write $t \sqsubseteq b$ if $(t \in P_i \implies b \in Q_i)$ for $1 \leq i \leq n$. Note that $t \sqsubseteq b$ is a necessary condition for the existence of a homomorphism from $\mathcal{T}$ to $\mathcal{B}$ that maps t to b.

For every node $t \in T$, let $H(t) \subseteq B$ be the set of all $b \in B$ such there is a homomorphism from $\mathcal{T}_t$ to $\mathcal{B}$ that maps t to b. The sets $H(t)$ can computed in the following way:

– If t is a leaf, then $H(t)$ consists of all $b \in B$ such that $t \sqsubseteq b$.
– If t has children $t_1, \dots, t_n$, we first compute $H(t_1), \dots, H(t_n)$ recursively. $H(t)$ consists of all $b \in B$ such that $t \sqsubseteq b$ and there exist $b_1 \in H(t_1), \dots, b_n \in H(t_n)$ such that $(b, b_1), \dots, (b, b_n) \in F$.

Then there is a homomorphism from $\mathcal{T}$ to $\mathcal{B}$ if and only if $H(r) \neq \emptyset$. Clearly, this yields a polynomial time algorithm.

The simple algorithmic idea underlying the example can be generalised from trees to structures that are, in some sense, "similar" to trees. It will be convenient to phrase the following definitions in terms of hypergraphs: A *hypergraph* is a pair $\mathcal{H} = (V, E)$ consisting of a finite set V of *vertices* and a set $E \subseteq 2^V$ of subsets of V called *(hyper)edges*. With each τ-structure $\mathcal{A}$ we associate a hypergraph $\mathcal{H}(\mathcal{A})$ as follows: The vertex set of $\mathcal{H}(\mathcal{A})$ is the universe of $\mathcal{A}$, and for all k, all k-ary $r \in \tau$, and all tuples $(a_1, \dots, a_k) \in r^{\mathcal{A}}$, the set $\{a_1, \dots, a_k\}$ is an edge of $\mathcal{H}(\mathcal{A})$. For a CSP-instance $\mathcal{I}$, we let $\mathcal{H}(\mathcal{I}) = \mathcal{H}(\mathcal{A}(\mathcal{I}))$. Note that the vertices of $\mathcal{H}(\mathcal{I})$ are the variables of $\mathcal{I}$ and the edges of $\mathcal{H}(\mathcal{I})$ are the scopes of the constraints of $\mathcal{I}$, where the *scope* of a constraint $Rx_1 \dots x_k$ is $\{x_1, \dots, x_k\}$.

A *tree decomposition* of a hypergraph $\mathcal{H} = (V, E)$ is a pair $(\mathcal{T}, B)$, where $\mathcal{T}$ is a tree and $B = (B_t)_{t \in T}$ a family of subsets of V such that for each $e \in E$ there is a node $t \in T$ with $e \subseteq B_t$, and for each $v \in V$ the set $\{t \in T \mid v \in B_t\}$ is connected in $\mathcal{T}$. The sets B_t are called the *bags* of the decomposition. The *width* of the decomposition (T, B) is $\max\{|B_t| \mid t \in T\} - 1$, and the *tree width* of $\mathcal{H}$, denoted by $\mathrm{tw}(\mathcal{H})$, is the minimum of the widths of all tree decompositions of $\mathcal{H}$. Figure 1 gives an example.

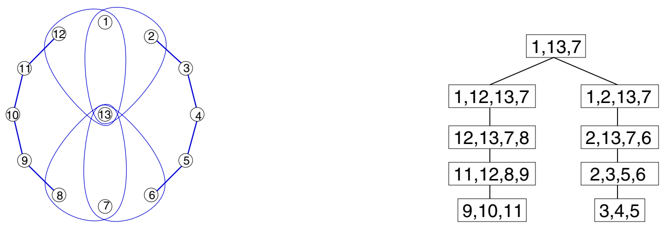


Fig. 1. A hypergraph $\mathcal{H}$ and a tree decomposition of $\mathcal{H}$ of width 3

The *tree width* $\mathrm{tw}(\mathcal{A})$ of a structure $\mathcal{A}$ is defined to be the tree width of its hypergraph $\mathcal{H}(\mathcal{A})$. We say that a class C of structures has *bounded tree width* if there is a k such that $\mathrm{tw}(\mathcal{A}) \leq k$ for all $\mathcal{A} \in \mathsf{C}$. (We shall use a similar terminology for other invariants such as bounded hypertree width later without explicitly defining it.)

It is NP-complete to decide whether a given hypergraph or structure has tree width k if k is given as part of the input [2]. However, for every fixed k there is a linear time algorithm that computes a tree decomposition of width k for a given structure $\mathcal{A}$ if the tree width of $\mathcal{A}$ is k [3].

Tree width may be seen as a measure for the "tree-likeness" of hypergraphs and structures. Freuder [18] generalised Example 11 and proved that for every class C of structures of bounded tree width, the problem $\textsc{Csp}(\mathsf{C}, -)$ is tractable. This can be further generalised: Two structures $\mathcal{A}$ and $\mathcal{A}'$ are *homomorphically equivalent* if there is a homomorphism from $\mathcal{A}$ to $\mathcal{A}'$ and a homomorphism from $\mathcal{A}'$ to $\mathcal{A}$. For example, all bipartite graphs with at least one edge are homomorphically equivalent. Observe that CSP-instances $\mathcal{I}$ and $\mathcal{I}'$ for which $\mathcal{A}(\mathcal{I})$ is homomorphically equivalent to $\mathcal{A}(\mathcal{I}')$ and $\mathcal{B}(\mathcal{I})$ is homomorphically equivalent to $\mathcal{B}(I')$ are either both satisfiable or both unsatisfiable.

Example 12. Let C be a class of structures such that each structure $\mathcal{A} \in \mathsf{C}$ is homomorphically equivalent to a labelled tree. Then $\textsc{Csp}(\mathsf{C}, -)$ is tractable.

To prove this, again we view $\textsc{Csp}(\mathsf{C}, -)$ as a homomorphism problem. The input consists of a structure $\mathcal{A} = (A, E, P_1, \dots, P_n)$ that is homomorphically equivalent to a labelled tree and a structure $\mathcal{B} = (B, F, Q_1, \dots, Q_n)$.

If we could efficiently compute a labelled tree $\mathcal{T}$ that is homomorphically equivalent to $\mathcal{A}$, then we could solve the problem by testing if there is a homomorphism from $\mathcal{T}$ to $\mathcal{B}$ as in Example 11. Unfortunately, there is no obvious way to find such a tree $\mathcal{T}$ (cf. [14]).

Let us define a game on $\mathcal{A}, \mathcal{B}$. The game is played by two players called *Spoiler* and *Duplicator*. Positions of the game are pairs $(a, b) \in A \times B$ such that $a \sqsubseteq b$ (i.e., $a \in P_i \Rightarrow b \in Q_i$ for $1 \leq i \leq n$). In the initial round of a play, Spoiler chooses an $a_0 \in A$ and Duplicator answers by choosing a $b_0 \in B$ with $b_0 \sqsupseteq a_0$. Then the initial position is (a_0, b_0). In each subsequent round, with current position (a, b), the next position (a', b') is determined as follows: Spoiler chooses a' such that $(a, a') \in E$. Then duplicator chooses b' such that $b' \sqsupseteq a'$ and $(b, b') \in F$. If in some round of the play Duplicator cannot answer, she loses. If she can continue to play forever, she wins.

We claim that there is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$ if and only if Duplicator has a winning strategy for the game. The forward direction is easy: If $h : A \to B$ is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$ then Duplicator simply answers every choice a of Spoiler by choosing $h(a)$. For the backward direction, suppose that Duplicator has a winning strategy for the game. We exploit the fact that $\mathcal{A}$ is homomorphically equivalent to a labelled tree. Assume first that $\mathcal{A}$ *is* a labelled tree. We define a mapping $h : A \to B$ as follows: Let a_0 be arbitrary. Suppose Spoiler chooses a_0 in the initial round. For every $a \in A$ there is precisely one path $a_0 a_1 \dots a_m = a$ from a_0 to a in the tree $\mathcal{A}$. We define $b_0, \dots, b_m$ to be the Duplicator's answer if the Spoiler plays $a_0, \ldots, a_m$ and let $h(a) = b_m$. It is easy to verify that h is indeed a homomorphism from $\mathcal{A}$ to $\mathcal{B}$.

Now suppose that $\mathcal{A}$ is not a tree. Let $\mathcal{T} = (T, E', P'_1, \ldots, P'_n)$ be a labelled tree that is homomorphically equivalent to $\mathcal{A}$, and let $g_1 : A \to T$, $g_2 : T \to A$ be homomorphisms from $\mathcal{A}$ to $\mathcal{T}$ and from $\mathcal{T}$ to $\mathcal{A}$, respectively. Then Duplicator has a winning strategy for the game on $\mathcal{T}$, $\mathcal{B}$: If Spoiler plays $t \in T$, she answers as she would have answered in the game on $\mathcal{A}, \mathcal{B}$ if Spoiler had played $g_2(t)$. Hence there is a homomorphism h from $\mathcal{T}$ to $\mathcal{B}$. Then $h \circ g_1$ is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$.

As the number of positions of the game is quadratic in the size of the input, it can be decided in polynomial time if Spoiler has a winning strategy for the game. Duplicator has a winning strategy if and only if Spoiler does not.

Again, this example can be generalised from trees to structures of bounded tree width. A class C of structures has *bounded tree width modulo homomorphic equivalence* if there is a k such that each structure $\mathcal{A} \in \mathsf{C}$ is homomorphically equivalent to a structure $\mathcal{A}'$ with $\mathrm{tw}(\mathcal{A}') \leq k$.

Theorem 13 (Dalmau, Kolaitis, and Vardi [14]). *Let* C *be a class of structures of bounded tree width modulo homomorphic equivalence. Then* $\textsc{Csp}(\mathsf{C}, -)$ *is tractable.*

Surprisingly, this theorem has a (partial) converse; for classes of structures of *bounded arity*, bounded tree width modulo homomorphic equivalence is also a necessary condition for the tractability of $\textsc{Csp}(\mathsf{C}, -)$. The *arity* of a structure is the maximum arity of its relations.

Theorem 14 (Grohe [20]). *Assume that* $\mathrm{FPT} \neq \mathrm{W}[1]$. *Let* C *be a recursively enumerable class of structures of bounded arity.*

Then $\textsc{Csp}(\mathsf{C}, -)$ *is tractable if and only if* C *has bounded tree width modulo homomorphic equivalence.*

Let us discuss the assumptions of this theorem: $\mathrm{FPT} \neq \mathrm{W}[1]$ is a complexity theoretic assumption from parameterized complexity theory (see [15,17]) that is widely believed to be true. The assumption that C be recursively enumerable is somewhat inessential. With a slightly stronger complexity theoretic assumption, the statement of the theorem can also be proved for classes C that are not recursively enumerable. The only serious restriction is that C be of bounded arity. While many natural CSPs (e.g., Clique, 3-Colourability) have bounded arity, some have not: Sat is one prominent example. The rest of the paper is devoted to $\textsc{Csp}(\mathsf{C}, -)$ for classes C of unbounded arity.

3.1 Unbounded Arity

Let $\mathcal{I}$ be a CSP-instance. Observe that the arity of the structure $\mathcal{A}(\mathcal{I})$ is the maximum edge size of the hypergraph $\mathcal{H}(\mathcal{I})$.

The following example shows that there are classes C of unbounded tree width modulo homomorphic equivalence (and thus unbounded arity) such that $\textsc{Csp}(\mathsf{C}, -)$ is tractable:

Example 15. For $n \geq 1$, let r_n be an n-ary relation symbol, and let $\mathcal{A}_n$ be the $\{r_n\}$-structure with universe $\{x_1, \ldots, x_n\}$ and $r_n^{\mathcal{A}} = \big\{(x_1, \ldots, x_n)\big\}$. Let $\mathsf{C} = \{\mathcal{A}_n \mid n \geq 1\}$. It is easy to see that the structure $\mathcal{A}_n$ has tree width $n-1$ and is not homomorphically equivalent to a structure of smaller tree width. Thus C has unbounded tree width modulo homomorphic equivalence.

But $\textsc{Csp}(\mathsf{C}, -)$ is tractable. To see this, let $\mathcal{I}$ be an instance of $\textsc{Csp}(\mathsf{C}, -)$, say, with $\mathcal{A}(\mathcal{I}) = \mathcal{A}_n$. Then $\mathcal{I}$ has a single constraint $R_n x_1 \ldots x_n$. Thus $\mathcal{I}$ is satisfiable if and only if R_n is nonempty, and clearly this can be checked in polynomial time.

At this point, it will be necessary to think about how CSP-instances are actually specified. The crucial question is how to specify the relations in the constraint language. In absence of any specific information about the instances, it seems most reasonable to just list the tuples of the relations explictly. Then the size of the representation of an instance $\mathcal{I}$ is roughly

$$||\mathcal{I}|| = |V| + |D| + \sum_{R \in L} |R| \cdot \text{arity}(R) + \sum_{c \in C} |c|,$$

where L denotes the constraint language of $\mathcal{I}$ and the length $|c|$ of a constraint $c = Rx_1 \ldots x_k$ is defined to be $k+1$. In the following, we call $||\mathcal{I}||$ the *size* of $\mathcal{I}$. Complexity will always be measured in terms of $||\mathcal{I}||$. Note that $||\mathcal{I}|| \geq |V| + |D| + |C|$. In general, $||\mathcal{I}||$ can be much larger than $|V| + |D| + |C|$. The best upper bound we get is $||\mathcal{I}|| = O(|V| + \ell \cdot |D|^{\ell} + \ell \cdot |C|)$, where ℓ is the maximum of the arities of the relations in the constraint language. For the bounded arity case, this means that $||\mathcal{I}||$ is polynomial in $|V| + |D| + |C|$, and thus in this case the choice of representation of the relations is not so significant.

Of course the explicit representation of the relations in the constraint language is not the only representation one can think of. Indeed, if relations of large arity occur in practice they are usually represented implicitly, for example, by the clauses of a $\textsc{Sat}$ instance. Note that a clause with n literals specifies a relation with $2^n - 1$ elements. The complexity of CSPs where the relations in the constraint language are represented implicitly (e.g. by clauses) is studied in [10]. But for the rest of this paper, we only consider the explicit representation.

Let $\mathcal{H} = (V, E)$ be a hypergraph. An *edge cover* of $\mathcal{H}$ is a set $C \subseteq E$ of edges such that $V = \bigcup C$. Here $\bigcup C = \bigcup_{e \in C} e = \{v \in V \mid \exists e \in C : \ v \in e\}$. The *edge cover number* of $\mathcal{H}$, denoted by $\rho(\mathcal{H})$, is the minimum cardinality of an edge cover of $\mathcal{H}$. As usually, the edge cover number of a structure is defined to be the edge cover number of its hypergraph. Note that the structure $\mathcal{A}_n$ of Example 15 has edge cover number 1 and tree width $n-1$.

Example 16. Let C be a class of structures of bounded edge cover number. Then $\textsc{Csp}(\mathsf{C}, -)$ is tractable. We leave it to the reader to verify this straightforward claim.

The observation of the previous example can be combined with the ideas developed for trees and structures of bounded tree width. Let $\mathcal{H} = (V, E)$ be a hypergraph. A *hypertree decomposition of* $\mathcal{H}$ is a triple $(\mathcal{T}, B, C)$, where $(\mathcal{T}, B)$ is a tree decomposition of $\mathcal{H}$ and $C = (C_t)_{t \in T}$ is a family of subsets of E such that for every $t \in V$ we have $B_t \subseteq \bigcup C_t$. The sets C_t are called the *guards* of the decomposition. Note that the guard C_t is an edge cover of the subhypergraph induced by the bag B_t.

The *width* of the decomposition (T, B, C) is $\max\{|C_t| \mid t \in T\}$. The *hypertree width* of $\mathcal{H}$, denoted by $\mathrm{hw}(\mathcal{H})$, is the minimum of the widths of the hypertree decompositions of H.

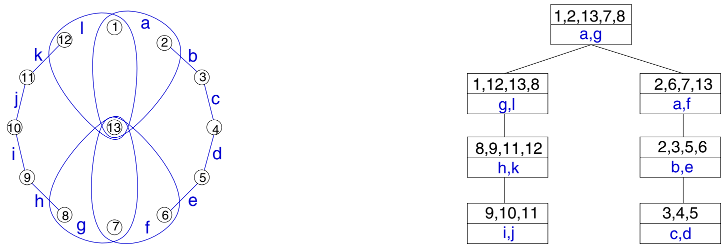


Fig. 2. A hypergraph $\mathcal{H}$ and a hypertree decomposition of $\mathcal{H}$ of width 2

Actually, what we call hypertree decomposition here is usually called "generalised hypertree decomposition" in the literature. The "standard" hypertree decompositions incorporate an additional technical condition on how the guards must be arranged in the tree. However, it has been proved in [1] that the width measures derived from the two notions are the same up to a constant factor, and for our purposes, this makes them interchangeable. Let us also remark that hypertree width is called "cover width" in [9]. Another related decomposition called "spread cut decomposition" has been introduced in [12].

As usually, we define the hypertree width of a structure to be the hypertree width of its hypergraph. Gottlob, Leone, and Scarcello [19] proved that $\mathrm{CSP}(\mathsf{C}, -)$ is tractable for all classes C of bounded hypertree width. Let us remark that Gottlob et al. stated their results for the problem of evaluating Boolean conjunctive database queries. Constraint satisfaction, homomorphism, Boolean conjunctive query evaluation, and also conjunctive query containment are all equivalent (see [27]). Chen and Dalmau generalised Gottlob et al.'s result to classes of structures of bounded hypertree width modulo homomorphic equivalence:

Theorem 17 (Chen and Dalmau [9]). *Let* C *be a class of structures of bounded hypertree width modulo homomorphic equivalence. Then* $\mathrm{CSP}(\mathsf{C}, -)$ *is tractable.*

But this is still not the end of the story. The problem of finding a minimum edge cover of a hypergraph $\mathcal{H} = (V, E)$ has the following integer linear programming (ILP) formulation:

$$\text{minimise} \sum_{e\in E} x_e \text{ subject to } \sum_{\substack{e\in E\\ \text{with } v\in e}} x_e \geq 1 \qquad \text{for all } v \in V,$$
$$x_e \in \{0,1\} \qquad \text{for all } e \in E.$$

Let us consider the linear programming relaxation of this ILP, where the integrality constraints $x_e \in \{0,1\}$ are replaced by the inequalities $x_e \geq 0$. A feasible solution for this linear program is called a *fractional edge cover* of $\mathcal{H}$. The weight $\sum_{e\in E} x_e$ of an optimal solution is called the *fractional edge cover number* of $\mathcal{H}$; it is denoted by $\rho^*(\mathcal{H})$. An optimal fractional edge cover and hence the fractional edge cover number can be computed in polynomial time by solving the linear program. Computing the (integral) edge cover number is NP-complete.

It can be shown (see, e.g., [30], where edge covers are called "set covers") that for every hypergraph $\mathcal{H}$ with n vertices,

$$\rho^*(\mathcal{H}) \leq \rho(\mathcal{H}) \leq \rho^*(\mathcal{H}) \cdot \ln n. \tag{1}$$

The following example shows that the upper bound is fairly tight:

Example 18. Let $\ell \geq 1$ and $\mathcal{H}$ be the following hypergraph: $\mathcal{H}$ has a vertex v_S for every subset S of $\{1,\dots,2\ell\}$ of cardinality ℓ. Furthermore, $\mathcal{H}$ has an edge $e_i = \{v_S \mid i \in S\}$ for every $i \in \{1,\dots,2\ell\}$.

Observe that the fractional edge cover number $\rho^*(\mathcal{H})$ is at most 2, because $x_e = 1/\ell$ for every edge e of $\mathcal{H}$ yields a fractional edge cover of weight 2. Actually, it is easy to see that $\rho^*(\mathcal{H}) = 2$. It is also easy to see that $\rho(\mathcal{H}) = \ell + 1$.

Theorem 19 (Grohe and Marx [21]). *Let C be a class of structures of bounded fractional edge cover number. Then* $\textsc{Csp}(\mathsf{C}, -)$ *is tractable.*

Observe that $\mathrm{hw}(\mathcal{H}) \leq \rho(\mathcal{H})$ for every hypergraph $\mathcal{H}$. To see this, consider the hypertree decomposition of $\mathcal{H}$ that consists of a one vertex tree, with a bag that contains all vertices and a guard that is an edge cover. Thus by (1) we have $\mathrm{hw}(\mathcal{H}) \leq \rho^*(\mathcal{H}) \cdot \log n$ for every n-vertex hypergraph. It can be shown that the hypertree width of the graph $\mathcal{H}$ of Example 18 is $\ell + 1 = \Theta(\log n)$ [21]. Conversely, the hypergraph that is the disjoint union of n edges of cardinality 1 has hypertree width 1 and fractional edge cover number n. Thus fractional edge cover number and hypertree width are incomparable. We can combine the two invariants as follows:

A *fractional hypertree decomposition* of a hypergraph $\mathcal{H} = (V, E)$ is a triple $(\mathcal{T}, B, g)$, where $(\mathcal{T}, B)$ is a tree decomposition of $\mathcal{H}$ and $g = (g_t)_{t\in T}$ is a family of functions from E to the nonnegative rationals such that for every $t \in T$ it holds that

$$\sum_{e \text{ with } v\in e} g_t(e) \geq 1 \quad \text{for all } v \in B_t$$

Hence the *guard* g_t is a fractional edge cover of the subhypergraph induced by the bag B_t. The *width* of the decomposition (T, B, C) is $\max\left\{ \sum_{e\in E} g_t(e) \;\middle|\; t \in T\right\}$. The *fractional hypertree width* of $\mathcal{H}$ is the minimum of the widths of the fractional hypertree decompositions of H.

Conjecture 3. *For every class* C *of structures,* $\textsc{Csp}(\mathsf{C}, -)$ *is tractable if and only if* C *has bounded fractional hypertree width modulo homomorphic equivalence.*

Both directions of this conjecture are open. The forward direction is implied by a technical conjecture stated in [21], which is concerned with the number of maximal subsets of the vertex set of a hypergraph that have a certain fractional edge cover number.

This is a good place to stop. All that remains to do is prove the conjectures.

Acknowledgements

The author is very grateful to Isolde Adler, Albert Atserias, Andrei Bulatov, Hubie Chen, Stephan Kreutzer, Andrei Krokhin, Daniel Marx, Nicole Schweikardt, and Marc Thurley for valuable comments and corrections on an earlier draft of this paper.

References

1. I. Adler, G. Gottlob, and M. Grohe. Hypertree-width and related hypergraph invariants. In S. Felsner, editor, *Proceedings of the 3rd European Conference on Combinatorics, Graph Theory, and Applications*, volume AE of *DMTCS Proceedings Series*, pages 5–10, 2005.
2. S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8:277–284, 1987.
3. H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
4. A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science*, pages 321–330, 2003.
5. A. Bulatov. H-coloring dichotomy revisited. *Theoretical Computer Science*, 349:31–39, 2005.
6. A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53:66–120, 2006.
7. A. Bulatov, A. Krokhin, and P. Jeavons. The complexity of maximal constraint languages. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 667–674, 2001.
8. A. Bulatov, A. Krokhin, and P. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.
9. H. Chen and V. Dalmau. Beyond hypertree width: Decomposition methods without decompositions. In P. van Beek, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, volume 3709 of *Lecture Notes in Computer Science*, pages 167–181. Springer-Verlag, 2005.
10. H. Chen and M. Grohe. Constraint satisfaction with succinctly specified relations. In preparation.
11. D. Cohen and P. Jeavons. The complexity of constraint languages. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 6. Elsevier, 2006.

12. D. Cohen, P. Jeavons, and M. Gyssens. A unified theory of structural tractability for constraint satisfaction and spread cut decomposition. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005. To appear.
13. V. Dalmau. Generalized majority-minority operations are tractable. In *Proceedings of the 20th IEEE Symposium on Logic in Computer Science*, pages 438–447, 2005.
14. V. Dalmau, Ph. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In P. Van Hentenryck, editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer-Verlag, 2002.
15. R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
16. T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1998.
17. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.
18. E.C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 4–9, 1990.
19. G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64:579–627, 2002.
20. M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 552–561, 2003.
21. M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proceedings of the of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 289–298, 2006.
22. P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990.
23. P. Hell, J. Nešetřil, and X. Zhu. Complexity of tree homomorphisms. *Discrete Applied Mathematics*, 70:23–36, 1996.
24. P. Hell, J. Nešetřil, and X. Zhu. Duality and polynomial testing of tree homomorphisms. *Transactions of the American Mathematical Society*, 348(4):1281–1297, 1996.
25. P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.
26. P. Jeavons, D. A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44(4):527–548, 1997.
27. Ph.G. Kolaitis and M.Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems*, pages 205–213, 1998.
28. R.E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.
29. T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing*, pages 216–226, 1978.
30. V.V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2004.

On the Representation of Kleene Algebras with Tests

Dexter Kozen

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA
kozen@cs.cornell.edu

Abstract. We investigate conditions under which a given Kleene algebra with tests is isomorphic to an algebra of binary relations. Two simple separation properties are identified that, along with star-continuity, are sufficient for nonstandard relational representation. An algebraic condition is identified that is necessary and sufficient for the construction to produce a standard representation.

1 Introduction

Kleene algebra with tests (KAT) is an equational system for program verification that combines Kleene algebra (KA), or the algebra of regular expressions, with Boolean algebra. One can model basic programming language constructs such as conditionals and while loops, verification conditions, and partial correctness assertions. KAT has been applied successfully in verification tasks involving communication protocols, source-to-source program transformation, concurrency control, compiler optimization, and dataflow analysis [1,2,3,4,5,6]. The system subsumes Hoare logic and is deductively complete for partial correctness over relational models [7].

There are many interesting and useful models of KAT: language-theoretic, relational, trace-based, matrix. In programming language semantics and verification, the relational models are of primary importance, because correctness conditions are often expressed as input/output conditions on the start and final state of the computation.

In relational models, actions and tests are represented as binary relations on some universal set of states. The class of all relational KATs is denoted REL. Because of the prominence of relational models in programming language semantics and verification, it is of interest to characterize them axiomatically or otherwise. It is known that REL satisfies no more equations than those satisfied by KATs in general, and the equational theory is *PSPACE*-complete [8]. This result extends to the *Hoare theory*, universal Horn formulas in which all premises are of the form $\mathsf{p} = 0$ [8,9]. However, the full Horn theories of REL and KAT diverge: the relationally valid Horn formula $\mathsf{p} \leq 1 \rightarrow \mathsf{p}^2 = \mathsf{p}$ is not true in all KATs or even in all star-continuous KATs. For example, it fails in the min,+ algebra or *tropical semiring* used in shortest path algorithms.

In this paper we explore conditions under which a Kleene algebra with tests can be represented isomorphically as a relational KAT. Not all algebras are so representable, even star-continuous ones; as observed above, the min,+ algebra is not. We have identified two basic first-order properties, Properties 1 and 2 below, that are sufficient for relational representation of idempotent semirings with tests, or Kleene algebras without

*. In the presence of *, these properties plus the infinitary star-continuity condition are sufficient for representation by a *nonstandard* relational model—one in which p^* is the least reflexive transitive relation containing p *in the algebra*, although not necessarily the set-theoretic reflexive transitive closure. We also identify a property that is equivalent to the assertion that the construction yields a standard model.

The two properties 1 and 2 can be viewed as *separation properties*. Essentially, they assert the existence of enough tests to allow binary relations to be characterized by their observable behavior, where the tests of the algebra are the observations. The two conditions are relatively weak, although for trivial reasons neither is a necessary condition for representation. We discuss the significance of Properties 1 and 2 further in Section 3 below.

The Stone representation theorem (see e.g. [10,11]) asserts that every Boolean algebra is isomorphic to a Boolean algebra of sets. After McKinsey's [12] and Tarski's [13] axiomatization of relation algebras, several authors [14,15,16] searched for a similar representation result for relation algebras but with only partial success. This work culminated in a counterexample of Lyndon [17]. In his conclusion, Lyndon discussed the possibility of a positive representation result in weaker systems. He mentioned specifically *relational rings*, which are essentially idempotent semirings or Kleene algebras without *. Work on the relational representation of dynamic algebra [18,19,20,21,22,23] built on this work and is analogous to the present results in the stronger setting in which all weakest preconditions are assumed to exist. The main result of this paper strengthens the representation results of [18,21] in that respect.

2 Preliminary Definitions

2.1 Kleene Algebra

Kleene algebra (KA) is the algebra of regular expressions [24,25]. The axiomatization used here is from [26]. A *Kleene algebra* is an algebraic structure $(K, +, \cdot, {}^*, 0, 1)$ that is an idempotent semiring under $+, \cdot, 0, 1$ such that $\mathsf{p}^*\mathsf{q}$ is the $\leq$-least solution to $\mathsf{q} + \mathsf{px} \leq \mathsf{x}$ and qp^* is the $\leq$-least solution to $\mathsf{q} + \mathsf{xp} \leq \mathsf{x}$. Here $\leq$ refers to the natural partial order on K: $\mathsf{p} \leq \mathsf{q} \overset{\text{def}}{\iff} \mathsf{p} + \mathsf{q} = \mathsf{q}$. This is a universal Horn axiomatization. A Kleene algebra is *star-continuous* if it satisfies the stronger infinitary property

$$\mathsf{pq}^*\mathsf{r} = \sup_n \mathsf{pq}^n\mathsf{r}. \tag{1}$$

The family of star-continuous Kleene algebras is denoted KA^*. It is a proper subclass of the Kleene algebras, but all naturally occurring Kleene algebras, including all relational models, are star-continuous.

The axioms for * say essentially that * behaves like the Kleene asterate operator of formal language theory or the reflexive transitive closure operator of relational algebra.

Standard models include the family of regular sets over a finite alphabet; the family of binary relations on a set; and the family of $n \times n$ matrices over another Kleene algebra. Other interpretations include the min,+ algebra or *tropical semiring* used in shortest path algorithms and models consisting of convex polyhedra used in computational geometry.

The completeness result of [26] says that all true identities between regular expressions interpreted as regular sets of strings are derivable from the axioms. In other words, the algebra of regular sets of strings over a finite alphabet P is the free Kleene algebra on generators P. The axioms are also complete for the equational theory of relational models.

2.2 Kleene Algebra with Tests

A *Kleene algebra with tests* (KAT) [5] is just a Kleene algebra with an embedded Boolean subalgebra. That is, it is a two-sorted structure $(K,\ B,\ +,\ \cdot,\ ^*,\ ^-,\ 0,\ 1)$ such that

- $(K,\ +,\ \cdot,\ ^*,\ 0,\ 1)$ is a Kleene algebra,
- $(B,\ +,\ \cdot,\ ^-,\ 0,\ 1)$ is a Boolean algebra, and
- $(B,\ +,\ \cdot,\ 0,\ 1)$ is a substructure of $(K,\ +,\ \cdot,\ 0,\ 1)$.

Elements of B are called *tests*. The Boolean complementation operator $^-$ is defined only on tests. We use the symbols $\mathsf{b}, \mathsf{c}, \mathsf{d}, \ldots$ to denote tests and $\mathsf{p}, \mathsf{q}, \mathsf{r}, \ldots$ to denote arbitrary elements of K.

The **while** program constructs are encoded as in propositional Dynamic Logic [27]:

$$\mathsf{p}\,;\,\mathsf{q} \overset{\text{def}}{=} \mathsf{pq}$$
$$\textbf{if } \mathsf{b} \textbf{ then } \mathsf{p} \textbf{ else } \mathsf{q} \overset{\text{def}}{=} \mathsf{bp} + \bar{\mathsf{b}}\mathsf{q}$$
$$\textbf{while } \mathsf{b} \textbf{ do } \mathsf{p} \overset{\text{def}}{=} (\mathsf{bp})^*\bar{\mathsf{b}}.$$

The Hoare partial correctness assertion $\{\mathsf{b}\}\ \mathsf{p}\ \{\mathsf{c}\}$ is expressed as the inequality $\mathsf{bp} \le \mathsf{pc}$ (equivalently, as the equation $\mathsf{bp}\bar{\mathsf{c}} = 0$ or the equation $\mathsf{bp} = \mathsf{bpc}$). All Hoare rules are derivable in KAT; indeed, KAT is deductively complete for relationally valid propositional Hoare-style rules involving partial correctness assertions [7] (propositional Hoare logic is not).

For A a set of tests, define $\bar{A} = \{\bar{\mathsf{b}} \mid \mathsf{b} \in A\}$ and $\sim A = B - A$. Note that $\bar{A}$ and $\sim A$ are not the same in general; however, they coincide if A is an ultrafilter or maximal ideal of B.

See [26,5,7,28] for a more detailed introduction to KA and KAT.

2.3 Relational Models

A *relational model* is a KAT whose elements are binary relations on some universal set U. The sequential composition operator $\cdot$ is interpreted as relational composition, the choice operator $+$ is interpreted as set-theoretic union, the iteration operator * is interpreted as reflexive transitive closure, the multiplicative identity 1 is interpreted as the identity relation on U, and the additive identity 0 is interpreted as the null relation. Tests are subsets of the identity relation on U, but not all subsets of the identity relation need be tests. The Boolean complementation operator on tests gives the set-theoretic complement in the identity relation.

A *nonstandard* relational model is the same, except that we do not require that p^* be the set-theoretic reflexive transitive closure, but only the $\leq$-least reflexive transitive relation containing p *in the algebra*.

The class of all relational KATs is denoted REL. If ϕ is a logical formula in the language of KAT, we write $\mathsf{REL} \vDash \phi$ and say that ϕ is *relationally valid* if it is true under all relational interpretations.

For a binary relation p on a set U, define the *domain* and *range* of p to be the sets

$$\mathsf{dom}(\mathsf{p}) \overset{\text{def}}{=} \{\mathsf{u} \mid \exists \mathsf{v}\ \ (\mathsf{u},\mathsf{v}) \in \mathsf{p}\} \qquad \mathsf{ran}(\mathsf{p}) \overset{\text{def}}{=} \{\mathsf{v} \mid \exists \mathsf{u}\ \ (\mathsf{u},\mathsf{v}) \in \mathsf{p}\},$$

respectively.

3 Representation

We will show that the following two natural properties, along with the star-continuity condition (1), are sufficient to construct a nonstandard relational representation. These properties are quite weak.

Property 1. $\mathsf{pq} = 0 \quad\Rightarrow\quad \exists \mathsf{b}\ \ \mathsf{p} = \mathsf{pb} \wedge \mathsf{q} = \bar{\mathsf{b}}\mathsf{q}$.

Property 2. $\mathsf{p} \not\leq \mathsf{q} \quad\Rightarrow\quad \exists \mathsf{b} \exists \mathsf{c}\ \ \mathsf{bpc} \neq 0 \wedge \mathsf{bqc} = 0$.

In relational models, Property 1 asserts that if pq vanishes, then there is a test that separates the range of p from the domain of q. This property is satisfied automatically in any system that postulates the existence of pre- and/or postconditions, such as dynamic algebra [19], Kleene algebra with domain and range operators [29], or Kleene modules [30]. It is related to expressibility conditions in Hoare logic [31] but somewhat weaker.

Property 2 asserts that actions can be distinguished by their interaction with tests. It is equivalent to the assertion that there exists no distinct inseparable pair, where the relation $\equiv$ of *inseparability* is defined by

$$\mathsf{p} \equiv \mathsf{q} \overset{\text{def}}{\Longleftrightarrow} \forall \mathsf{b} \forall \mathsf{c}\ (\mathsf{bpc} = 0 \Leftrightarrow \mathsf{bqc} = 0).$$

The significance of this requirement is captured in the following proposition.

Proposition 1. *Let* p *and* q *be terms in the language of* KAT, *and let* b *and* c *be test variables not occurring in* p *or* q. *The following are equivalent:*

(i) $\mathsf{p} \leq \mathsf{q}$ *is valid,*
(ii) $\mathsf{bqc} = 0 \to \mathsf{bpc} = 0$ *is valid,*
(iii) $\mathsf{bqc} = 0 \to \mathsf{bpc} = 0$ *is relationally valid.*

Proof. The implications (i) $\Rightarrow$ (ii) and (ii) $\Rightarrow$ (iii) are trivial. For (iii) $\Rightarrow$ (i), if $\mathsf{p} \leq \mathsf{q}$ is not valid, then it fails in some relational model, since the equational theories of KAT and REL are the same. Let $(s,t) \in \mathsf{p} - \mathsf{q}$ in this model, and reinterpret b as $\{s\}$ and c as $\{t\}$. Then bpc contains (s,t), but $\mathsf{bqc} = 0$.

Let K, B be a KAT satisfying Properties 1 and 2. As with all Stone-like constructions, our universal set of states will be the set of ultrafilters of B.

Recall that a *filter* is a subset F of B such that

(i) $\mathsf{bc} \in F \;\Leftrightarrow\; \mathsf{b} \in F$ and $\mathsf{c} \in F$
(ii) $1 \in F$
(iii) $0 \notin F$,

and an *ultrafilter* is a maximal filter. Dually, an *ideal* is a subset I of B such that

(i) $\mathsf{b} + \mathsf{c} \in I \;\Leftrightarrow\; \mathsf{b} \in I$ and $\mathsf{c} \in I$
(ii) $1 \notin I$
(iii) $0 \in I$.

Ideals are the kernels of Boolean algebra homomorphisms. Note that F is a filter iff $\bar{F}$ is an ideal, and u is an ultrafilter iff $\bar{u}$ is a maximal ideal. By Zorn's lemma, every filter is contained in an ultrafilter and every ideal is contained in a maximal ideal. Every ultrafilter u satisfies the property that for all b, either $\mathsf{b} \in u$ or $\bar{\mathsf{b}} \in u$, but not both; therefore $\sim u = \bar{u}$.

Let U denote the set of ultrafilters of B. The well-known Stone construction (see e.g. [10,11]) produces a Boolean algebra isomorphic to B whose elements are subsets of U and whose Boolean operations are the usual set-theoretic ones. The subset corresponding to b is $\mathsf{b}' \overset{\text{def}}{=} \{u \mid \mathsf{b} \in u\}$. We denote this set-theoretic Boolean algebra by B'.

A *coideal* of a Boolean algebra B is a complement of an ideal; that is, it is a subset C of B satisfying

(i) $\mathsf{b} + \mathsf{c} \in C \;\Leftrightarrow\; \mathsf{b} \in C$ or $\mathsf{c} \in C$
(ii) $1 \in C$
(iii) $0 \notin C$.

Lemma 1. *Every coideal contains an ultrafilter as a subset.*

Proof. If C is a coideal, then $\sim C$ is an ideal. By Zorn's Lemma, $\sim C$ extends to a maximal ideal M. Then $\sim M$ is an ultrafilter and is a subset of C.

Definition 1.

$$\begin{aligned}
\mathsf{In}(\mathsf{p},\mathsf{q}) &\overset{\text{def}}{=} \{\mathsf{d} \mid \mathsf{pdq} \neq 0\} \\
\mathsf{Pre}(\mathsf{p}) &\overset{\text{def}}{=} \{\mathsf{b} \mid \mathsf{bp} \neq 0\} \;=\; \mathsf{In}(1,\mathsf{p}) \\
\mathsf{Post}(\mathsf{p}) &\overset{\text{def}}{=} \{\mathsf{c} \mid \mathsf{pc} \neq 0\} \;=\; \mathsf{In}(\mathsf{p},1).
\end{aligned}$$

Lemma 2. *If* $\mathsf{pq} \neq 0$*, then* $\mathsf{In}(\mathsf{p},\mathsf{q})$ *is a coideal. If* $\mathsf{p} \neq 0$*, then* $\mathsf{Pre}(\mathsf{p})$ *and* $\mathsf{Post}(\mathsf{p})$ *are coideals.*

Proof. For $\mathsf{In}(\mathsf{p},\mathsf{q})$,

(i) $\mathsf{p}(\mathsf{c}+\mathsf{d})\mathsf{q} \neq 0 \;\Leftrightarrow\; \mathsf{pcq} + \mathsf{pdq} \neq 0 \;\Leftrightarrow\; \mathsf{pcq} \neq 0$ or $\mathsf{pdq} \neq 0$,
(ii) $\mathsf{pq} \neq 0 \;\Rightarrow\; \mathsf{p1q} \neq 0 \;\Rightarrow\; 1 \in \mathsf{In}(\mathsf{p},\mathsf{q})$,
(iii) $\mathsf{p0q} = 0 \;\Rightarrow\; 0 \notin \mathsf{In}(\mathsf{p},\mathsf{q})$.

$\mathsf{Pre}(\mathsf{p})$ and $\mathsf{Post}(\mathsf{p})$ are special cases.

A collection $\{C_\alpha\}$ of coideals is *downward-directed* if for any pair C_α and C_β, there is a C_γ with $C_\gamma \subseteq C_\alpha \cap C_\beta$.

Lemma 3. *The intersection of any downward-directed set of coideals is a coideal.*

Proof. Equivalently, the union of any upward-directed set of ideals is an ideal. The three conditions are easily checked.

Lemma 4.

(i) *If* $\mathsf{p}_1 \le \mathsf{p}_2$ *and* $\mathsf{q}_1 \le \mathsf{q}_2$, *then* $\mathsf{In}(\mathsf{p}_1, \mathsf{q}_1) \subseteq \mathsf{In}(\mathsf{p}_2, \mathsf{q}_2)$.
(ii) *If* $\mathsf{p}_1 \le \mathsf{p}_2$, *then* $\mathsf{Post}(\mathsf{p}_1) \subseteq \mathsf{Post}(\mathsf{p}_2)$.
(iii) *If* $\mathsf{p}_1 \le \mathsf{p}_2$, *then* $\mathsf{Pre}(\mathsf{p}_1) \subseteq \mathsf{Pre}(\mathsf{p}_2)$.

Proof. Statement (i) follows easily from the definitions. Statements (ii) and (iii) are special cases.

We now define our relational model R. For all $\mathsf{p} \in K$, define

Definition 2.

$$\mathsf{p}^R \overset{\text{def}}{=} \{(u,v) \mid \forall \mathsf{b} \in u \ \ \forall \mathsf{c} \in v \ \ \mathsf{bpc} \neq 0\}.$$

Since $\mathsf{bp} = 0$ iff $\mathsf{p} = \bar{\mathsf{b}}\mathsf{p}$ (see e.g. [7, Section 3]), we have the following facts:

$$\begin{aligned} \mathsf{Pre}(\mathsf{p}) = \{\mathsf{b} \mid \mathsf{bp} \neq 0\} &= \{\mathsf{b} \mid \mathsf{p} \neq \bar{\mathsf{b}}\mathsf{p}\} \\ {\sim}\mathsf{Pre}(\mathsf{p}) = \{\mathsf{b} \mid \mathsf{bp} = 0\} &= \{\mathsf{b} \mid \mathsf{p} = \bar{\mathsf{b}}\mathsf{p}\} \\ {\sim}\overline{\mathsf{Pre}}(\mathsf{p}) = \{\bar{\mathsf{b}} \mid \mathsf{bp} = 0\} &= \{\bar{\mathsf{b}} \mid \mathsf{p} = \bar{\mathsf{b}}\mathsf{p}\}. \end{aligned}$$

If $\mathsf{p} \neq 0$, then $\mathsf{Pre}(\mathsf{p})$ is a coideal, ${\sim}\mathsf{Pre}(\mathsf{p})$ is an ideal, and ${\sim}\overline{\mathsf{Pre}}(\mathsf{p})$ is a filter.

Lemma 5. *Let u be an ultrafilter. The following are equivalent:*

(i) $u \subseteq \mathsf{Pre}(\mathsf{p})$
(ii) ${\sim}\overline{\mathsf{Pre}}(\mathsf{p}) \subseteq u$
(iii) $u \in \mathsf{dom}(\mathsf{p}^R)$.

Proof. (i) $\Leftrightarrow$ (ii): ${\sim}\overline{\mathsf{Pre}}(\mathsf{p}) \subseteq u$ iff ${\sim}\mathsf{Pre}(\mathsf{p}) \subseteq \bar{u}$ iff ${\sim}\bar{u} \subseteq \mathsf{Pre}(\mathsf{p})$ iff $u \subseteq \mathsf{Pre}(\mathsf{p})$, since $u = {\sim}\bar{u}$.

(iii) $\Rightarrow$ (i): If $(u,v) \in \mathsf{p}^R$, then for all $\mathsf{b} \in u$ and $\mathsf{c} \in v$, $\mathsf{bpc} \neq 0$. Since $1 \in v$, we have that for all $\mathsf{b} \in u$, $\mathsf{bp} \neq 0$. Then $u \subseteq \mathsf{Pre}(\mathsf{p})$ by definition of $\mathsf{Pre}(\mathsf{p})$.

(i) $\Rightarrow$ (iii): If $u \subseteq \mathsf{Pre}(\mathsf{p})$, then $\mathsf{bp} \neq 0$ for all $\mathsf{b} \in u$. By Lemma 2, for all $\mathsf{b} \in u$, $\mathsf{Post}(\mathsf{bp})$ is a coideal. By Lemmas 3 and 4(ii), $\bigcap_{\mathsf{b} \in u} \mathsf{Post}(\mathsf{bp})$ is a coideal, therefore contains an ultrafilter v by Lemma 1. Thus for all $\mathsf{b} \in u$ and $\mathsf{c} \in v$, $\mathsf{c} \in \mathsf{Post}(\mathsf{bp})$, therefore $\mathsf{bpc} \neq 0$ and $(u,v) \in \mathsf{p}^R$.

The following is the main theorem of this section.

Theorem 1. *Let K, B be a star-continuous* KAT *satisfying Properties 1 and 2. The set $\{\mathsf{p}^R \mid \mathsf{p} \in \mathsf{K}\}$ is a nonstandard relational* KAT *with tests $\{\mathsf{b}^R \mid \mathsf{b} \in \mathsf{B}\}$, and the map $\mathsf{p} \mapsto \mathsf{p}^R$ is a* KAT *isomorphism.*

Proof. If $\mathsf{p} \le \mathsf{q}$ then $\mathsf{p}^R \subseteq \mathsf{q}^R$, since $\mathsf{p} \le \mathsf{q} \Rightarrow \mathsf{bpc} \le \mathsf{bqc}$. Thus the map $\mathsf{p} \mapsto \mathsf{p}^R$ is monotone.

To show that $\mathsf{p} \mapsto \mathsf{p}^R$ is one-to-one, it suffices to show that if $\mathsf{p} \not\leq \mathsf{q}$, then $\mathsf{p}^R \not\subseteq \mathsf{q}^R$. Suppose that $\mathsf{p} \not\leq \mathsf{q}$. By Property 2, there exist b and c such that $\mathsf{bpc} \neq 0$ and $\mathsf{bqc} = 0$. By Lemma 2, $\mathsf{Pre(bpc)} = \{\mathsf{d} \mid \mathsf{dbpc} \neq 0\}$ is a coideal, therefore contains an ultrafilter u by Lemma 1. By definition of $\mathsf{Pre(bpc)}$, we have $\mathsf{dbpc} \neq 0$ for all $\mathsf{d} \in u$. It follows that $\mathsf{b} \in u$, since either $\mathsf{b} \in u$ or $\overline{\mathsf{b}} \in u$, and the latter is impossible. Moreover, by Lemma 2, for all $\mathsf{d} \in u$, $\mathsf{Post(dbpc)} = \{\mathsf{e} \mid \mathsf{dbpce} \neq 0\}$ is a coideal. In addition, it follows from Lemma 4(ii) that the set $\{\mathsf{Post(dbpc)} \mid \mathsf{d} \in u\}$ is downward-directed, since

$$\mathsf{Post}(\mathsf{d}_1\mathsf{d}_2\mathsf{bpc}) \subseteq \mathsf{Post}(\mathsf{d}_1\mathsf{bpc}) \cap \mathsf{Post}(\mathsf{d}_2\mathsf{bpc}).$$

By Lemma 3, $\bigcap_{\mathsf{d}\in u} \mathsf{Post(dbpc)}$ is a coideal, therefore contains an ultrafilter v by Lemma 1. As with u, $\mathsf{c} \in v$. Then $(u, v) \in \mathsf{p}^R$, but $(u, v) \notin \mathsf{q}^R$ since $\mathsf{bqc} = 0$.

Next we show that $\mathsf{p} \mapsto \mathsf{p}^R$ is a homomorphism with respect to addition; that is, $(\mathsf{p} + \mathsf{q})^R = \mathsf{p}^R \cup \mathsf{q}^R$. The reverse inclusion follows from monotonicity. For the forward inclusion, suppose $(u, v) \notin \mathsf{p}^R \cup \mathsf{q}^R$. Then there exist $\mathsf{b}_1, \mathsf{b}_2 \in u$ and $\mathsf{c}_1, \mathsf{c}_2 \in v$ such that $\mathsf{b}_1\mathsf{pc}_1 = 0$ and $\mathsf{b}_2\mathsf{qc}_2 = 0$. Since u and v are filters, $\mathsf{b}_1\mathsf{b}_2 \in u$ and $\mathsf{c}_1\mathsf{c}_2 \in v$, and $\mathsf{b}_1\mathsf{b}_2\mathsf{pc}_1\mathsf{c}_2 = 0$ and $\mathsf{b}_1\mathsf{b}_2\mathsf{qc}_1\mathsf{c}_2 = 0$. Then $\mathsf{b}_1\mathsf{b}_2(\mathsf{p} + \mathsf{q})\mathsf{c}_1\mathsf{c}_2 = 0$, so $(u, v) \notin (\mathsf{p} + \mathsf{q})^R$.

Next we show that $\mathsf{p} \mapsto \mathsf{p}^R$ is a homomorphism with respect to multiplication; that is, $(\mathsf{pq})^R = \mathsf{p}^R \circ \mathsf{q}^R$. For the forward inclusion, suppose $(u, v) \in (\mathsf{pq})^R$. Then for all $\mathsf{b} \in u$ and $\mathsf{c} \in v$, $\mathsf{bpqc} \neq 0$. By Lemma 2, for all $\mathsf{b} \in u$ and $\mathsf{c} \in v$, $\mathsf{In(bp, qc)}$ is a coideal. Moreover, the set $\{\mathsf{In(bp, qc)} \mid \mathsf{b} \in u,\ \mathsf{c} \in v\}$ is downward-directed, since

$$\mathsf{In}(\mathsf{b}_1\mathsf{b}_2\mathsf{p}, \mathsf{qc}_1\mathsf{c}_2) \subseteq \mathsf{In}(\mathsf{b}_1\mathsf{p}, \mathsf{qc}_1) \cap \mathsf{In}(\mathsf{b}_2\mathsf{p}, \mathsf{qc}_2).$$

By Lemma 3, $\bigcap\{\mathsf{In(bp, qc)} \mid \mathsf{b} \in u,\ \mathsf{c} \in v\}$ is a coideal, therefore contains an ultrafilter w by Lemma 1. Then for all $\mathsf{b} \in u$, $\mathsf{c} \in v$, and $\mathsf{d} \in w$, $\mathsf{bpdqc} \neq 0$, therefore $\mathsf{bpd} \neq 0$ and $\mathsf{dqc} \neq 0$. It follows that $(u, w) \in \mathsf{p}^R$ and $(w, v) \in \mathsf{q}^R$, therefore $(u, v) \in \mathsf{p}^R \circ \mathsf{q}^R$.

For the reverse inclusion, we need Property 1. Suppose $(u, w) \in \mathsf{p}^R$ and $(w, v) \in \mathsf{q}^R$. Then for all $\mathsf{b} \in u$, $\mathsf{c} \in v$, and $\mathsf{d} \in w$, we have $\mathsf{bpd} \neq 0$ and $\mathsf{dqc} \neq 0$. If $\mathsf{bpqc} = 0$ for some $\mathsf{b} \in u$ and $\mathsf{c} \in v$, then by Property 1, there exists d such that $\mathsf{bp} = \mathsf{bpd}$ and $\mathsf{qc} = \overline{\mathsf{d}}\mathsf{qc}$. Either $\mathsf{d} \in w$ or $\overline{\mathsf{d}} \in w$. If the former, then $\mathsf{dqc} = \mathsf{d}\overline{\mathsf{d}}\mathsf{qc} = 0$. If the latter, then $\mathsf{bp}\overline{\mathsf{d}} = \mathsf{bpd}\overline{\mathsf{d}} = 0$. In either case, we have a contradiction. Thus for all $\mathsf{b} \in u$ and $\mathsf{c} \in v$, $\mathsf{bpqc} \neq 0$, therefore $(u, v) \in (\mathsf{pq})^R$.

For tests, we must show that $\mathsf{b}^R = \{(u, u) \mid \mathsf{b} \in u\}$, that 1^R is the identity relation on U, that 0^R is the empty relation, and that the map $\mathsf{b} \mapsto \mathsf{b}^R$ is a homomorphism with respect to negation. We have

$$\mathsf{b}^R = \{(u, v) \mid \forall \mathsf{c} \in u\ \forall \mathsf{d} \in v\ \ \mathsf{cbd} \neq 0\}.$$

Thus $(u, v) \in \mathsf{b}^R$ iff $u = v$ and $\mathsf{b} \in u$, therefore $\mathsf{b}^R = \{(u, u) \mid \mathsf{b} \in u\}$. In particular, $1^R = \{(u, u) \mid 1 \in u\}$, the identity relation, and $0^R = \{(u, u) \mid 0 \in u\} = \emptyset$. Since $\mathsf{b} \in u$ iff $\overline{\mathsf{b}} \notin u$,

$$\overline{\mathsf{b}}^R = \{(u, u) \mid \overline{\mathsf{b}} \in u\} \quad = \quad \{(u, u) \mid \mathsf{b} \notin u\} \quad = \quad 1^R - \mathsf{b}^R.$$

Finally, for *, it follows from the star-continuity of K, B and the fact that the map $\mathsf{p} \mapsto \mathsf{p}^R$ is an order isomorphism that $\mathsf{q}^{*R} = \sup_n (\mathsf{q}^R)^n$.

4 Star

In this section we identify an algebraic condition (Condition 2 below) under which the construction of Section 3 yields a standard relational model. This occurs exactly when $\mathsf{q}^{R*} = \mathsf{q}^{*R}$ for all q, where q^{R*} denotes the set-theoretic reflexive transitive closure of q^R and q^{*R} is the representation of q^* in R.

Endow U with the Stone topology generated by B' and $U \times U$ with the product topology. The basic open sets of these spaces are sets of the form b' and $\mathsf{b}' \times \mathsf{c}'$, respectively. Let $\mathrm{cl}(A)$ denote the closure of A in either topology.

Lemma 6. *Every* p^R *is closed in* $U \times U$.

Proof. If $(u, v) \notin \mathsf{p}^R$, then there exist $\mathsf{b} \in u$ and $\mathsf{c} \in v$ such that $\mathsf{bpc} = 0$. Then $(u, v) \in \mathsf{b}' \times \mathsf{c}'$ and

$$(\mathsf{b}' \times \mathsf{c}') \cap \mathsf{p}^R = (\mathsf{bpc})^R \ = \ \emptyset.$$

Thus $\mathsf{b}' \times \mathsf{c}'$ is a basic open neighborhood of (u, v) disjoint from p^R. Since $(u, v) \notin \mathsf{p}^R$ was arbitrary, $(U \times U) - \mathsf{p}^R$ is open, therefore p^R is closed.

Lemma 7. *The sets* $\mathsf{dom}(\mathsf{p}^R)$ *and* $\mathsf{ran}(\mathsf{p}^R)$ *are closed in* U.

Proof. By Lemma 5, $\mathsf{dom}(\mathsf{p}^R)$ is the set of ultrafilters u extending the filter $\sim\overline{\mathsf{Pre}}(\mathsf{p})$. But the set of ultrafilters extending any filter F is closed, since it is the intersection of basic closed sets:

$$\{u \mid F \subseteq u\} = \bigcap_{\mathsf{b} \in F} \{u \mid \mathsf{b} \in u\} \ = \ \bigcap_{\mathsf{b} \in F} \mathsf{b}'.$$

The argument for $\mathsf{ran}(\mathsf{p})$ is symmetric.

Lemma 8. *Let* $\{\mathsf{q}_\alpha\}$ *be a collection of elements of* K *and* p *an element of* K *such that for all* $\mathsf{b}, \mathsf{c} \in B$, $\mathsf{bpc} = \sup_\alpha \mathsf{bq}_\alpha\mathsf{c}$. *Then* $\mathsf{p}^R = \mathrm{cl}(\bigcup_\alpha \mathsf{q}_\alpha^R)$. *In particular,* $\mathsf{q}^{*R} = \mathrm{cl}(\mathsf{q}^{R*})$.

Proof. The inclusion $\supseteq$ holds by Lemma 6. If $(u, v) \in \mathsf{p}^R - \mathrm{cl}(\bigcup_\alpha \mathsf{q}_\alpha^R)$, then there exists a basic open neighborhood $\mathsf{b}' \times \mathsf{c}'$ of (u, v) disjoint from $\bigcup_\alpha \mathsf{q}_\alpha^R$. Then $(\mathsf{bq}_\alpha\mathsf{c})^R = (\mathsf{b}' \times \mathsf{c}') \cap \mathsf{q}_\alpha^R = \emptyset$ for all α, thus $\mathsf{bq}_\alpha\mathsf{c} = 0$ for all α, therefore $\sup_\alpha \mathsf{bq}_\alpha\mathsf{c} = 0$. But $(u, v) \in (\mathsf{b}' \times \mathsf{c}') \cap \mathsf{p}^R = (\mathsf{bpc})^R$, thus $(\mathsf{bpc})^R \neq \emptyset$ and $\mathsf{bpc} \neq 0$. This contradicts the assumption.

A necessary and sufficient condition for the relational model constructed in Section 3 to be standard is the following *uniform halting condition*:

Condition 2.

$$\forall n\, \exists \mathsf{b} \in u\, \exists \mathsf{c} \in v\ \ \mathsf{bq}^n\mathsf{c} = 0 \Rightarrow \exists \mathsf{b} \in u\, \exists \mathsf{c} \in v\, \forall n\ \ \mathsf{bq}^n\mathsf{c} = 0.$$

Condition 2 says that if for each n there are properties of (u, v) that cause it not to be an input/output pair of the program q^n, then there is a pair of such properties that work uniformly over all n. Intuitively, the input/output relation of a loop depends on only finitely many testable properties. Equivalently,

$$\forall \mathsf{b} \in u\, \forall \mathsf{c} \in v\, \exists n\ \ \mathsf{bq}^n\mathsf{c} \neq 0 \Rightarrow \exists n\, \forall \mathsf{b} \in u\, \forall \mathsf{c} \in v\ \ \mathsf{bq}^n\mathsf{c} \neq 0.$$

Theorem 3. *Condition 2 is equivalent to the property* $\mathsf{q}^{*R} = \mathsf{q}^{R*}$.

Proof. The left-hand side of Condition 2 says exactly that $(u,v) \notin \mathsf{q}^{R^n}$ for all n, thus $(u,v) \notin \mathsf{q}^{R*}$. We also have

$$\bigcup_n (\mathsf{b}\mathsf{q}^n\mathsf{c})^R = \bigcup_n (\mathsf{b}' \times \mathsf{c}') \cap \mathsf{q}^{nR} \;=\; (\mathsf{b}' \times \mathsf{c}') \cap \bigcup_n \mathsf{q}^{nR} \;=\; (\mathsf{b}' \times \mathsf{c}') \cap \mathsf{q}^{R*},$$

so that the right-hand side of Condition 2 can be rewritten

$$\begin{aligned}
&\exists \mathsf{b} \in u\, \exists \mathsf{c} \in v\, \forall n\;\; \mathsf{b}\mathsf{q}^n\mathsf{c} = 0 \\
&\Leftrightarrow \exists \mathsf{b}\, \exists \mathsf{c}\;\; (u,v) \in \mathsf{b}' \times \mathsf{c}' \;\text{ and }\; \forall n\, (\mathsf{b}\mathsf{q}^n\mathsf{c})^R = \emptyset \\
&\Leftrightarrow \exists \mathsf{b}\, \exists \mathsf{c}\;\; (u,v) \in \mathsf{b}' \times \mathsf{c}' \;\text{ and }\; \bigcup_n (\mathsf{b}\mathsf{q}^n\mathsf{c})^R = \emptyset \\
&\Leftrightarrow \exists \mathsf{b}\, \exists \mathsf{c}\;\; (u,v) \in \mathsf{b}' \times \mathsf{c}' \;\text{ and }\; (\mathsf{b}' \times \mathsf{c}') \cap \mathsf{q}^{R*} = \emptyset.
\end{aligned}$$

This says exactly that there is a basic open neighborhood of (u,v) disjoint from q^{R*}. Thus the implication of Condition 2 says exactly that $\sim\!\mathsf{q}^{R*}$ is open; that is, q^{R*} is closed. By Lemma 8, q^{*R} is the closure of q^{R*}, therefore they are equal if and only if q^{R*} is closed.

5 Open Problems

Theorem 3 does not say that a standard representation does not exist if Condition 2 fails. In the case of countable K, some variant of the Tarski–Rasiowa–Sikorski lemma or the Baire category theorem might be used to drop out nonstandard points from the model constructed above, giving a standard relational model or perhaps a homomorphic image of one; see [19,21]. The constructions of [19,21] do not seem to apply directly. On the other hand, neither do we have a negative result. The counterexamples of [20,22] for dynamic algebra do not immediately provide counterexamples for KAT, but the counterexample of [20] may be adaptable. Such questions remain for future investigation.

Axiomatization of the universal Horn theory of relational models is another interesting open question. This theory is an extension of the universal Horn theory of the star-continuous Kleene algebras, and both theories are known to be Π^1_1-complete [32,33], therefore not finitely axiomatizable. However, the star-continuous algebras have a succinct infinitary axiomatization containing a single infinitary rule (1). It is interesting to ask whether the relational algebras have a finitary axiomatization relative to this. Presumably the Horn formula $\mathsf{p} \le 1 \to \mathsf{p}^2 = \mathsf{p}$ would be a candidate axiom. Considerable progress in this direction has been made by Hardin [34].

Acknowledgements

Thanks to Samson Abramsky for valuable comments. It was he who suggested the term “coideal”. This work was supported in part by NSF grant CCR-0105586 and by ONR Grant N00014-01-1-0968. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the US Government.

References

1. Angus, A., Kozen, D.: Kleene algebra with tests and program schematology. Technical Report 2001-1844, Computer Science Department, Cornell University (2001)
2. Barth, A., Kozen, D.: Equational verification of cache blocking in LU decomposition using Kleene algebra with tests. Technical Report 2002-1865, Computer Science Department, Cornell University (2002)
3. Cohen, E.: Lazy caching in Kleene algebra (1994) `http://citeseer.nj.nec.com/22581.html`.
4. Cohen, E.: Hypotheses in Kleene algebra. Technical Report TM-ARH-023814, Bellcore (1993) `http://citeseer.nj.nec.com/1688.html`.
5. Kozen, D.: Kleene algebra with tests. Transactions on Programming Languages and Systems **19**(3) (1997) 427–443
6. Kozen, D., Patron, M.C.: Certification of compiler optimizations using Kleene algebra with tests. In Lloyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palamidessi, C., Pereira, L.M., Sagiv, Y., Stuckey, P.J., eds.: Proc. 1st Int. Conf. Computational Logic (CL2000). Volume 1861 of Lecture Notes in Artificial Intelligence., London, Springer-Verlag (2000) 568–582
7. Kozen, D.: On Hoare logic and Kleene algebra with tests. Trans. Computational Logic **1**(1) (2000) 60–76
8. Cohen, E., Kozen, D., Smith, F.: The complexity of Kleene algebra with tests. Technical Report 96-1598, Computer Science Department, Cornell University (1996)
9. Kozen, D., Smith, F.: Kleene algebra with tests: Completeness and decidability. In van Dalen, D., Bezem, M., eds.: Proc. 10th Int. Workshop Computer Science Logic (CSL'96). Volume 1258 of Lecture Notes in Computer Science., Utrecht, The Netherlands, Springer-Verlag (1996) 244–259
10. Bell, J., Slomson, A.: Models and Ultraproducts. North Holland (1971)
11. Halmos, P.: Lectures on Boolean Algebras. Springer Verlag (1974)
12. McKinsey, J.: Postulates for the calculus of binary relations. J. Symb. Logic **5**(3) (1940) 85–97
13. Tarski, A.: On the calculus of relations. J. Symb. Logic **6**(3) (1941) 73–89
14. Everett, C., Ulam, S.: Projective algebra I. Amer. J. Math. **68**(1) (1946) 77–88
15. Jonsson, B., Tarski, A.: Representation problems for relation algebras. Bull. Amer. Math. Soc. **54** (1948) 80 abstract 89t.
16. McKinsey, J.: On the representation of projective algebras. Amer. J. Math. **70** (1948) 375–384
17. Lyndon, R.: The representation of relation algebras. Ann. Math. **51**(3) (1950) 707–729
18. Kozen, D.: On the representation of dynamic algebras. Technical Report RC7898, IBM Thomas J. Watson Research Center (1979)
19. Kozen, D.: On the duality of dynamic algebras and Kripke models. In Engeler, E., ed.: Proc. Workshop on Logic of Programs. Volume 125 of Lecture Notes in Computer Science., Springer-Verlag (1979) 1–11
20. Kozen, D.: On the representation of dynamic algebras II. Technical Report RC8290, IBM Thomas J. Watson Research Center (1980)
21. Kozen, D.: A representation theorem for models of *-free *PDL*. In: Proc. 7th Colloq. Automata, Languages, and Programming, EATCS (1980) 351–362
22. Reiterman, J., Trnková, V.: Dynamic algebras which are not Kripke structures. In: Proc. 9th Symp. Math. Found. of Computer Science (MFCS'80). (1980) 528–538
23. Németi, I.: Every free algebra in the variety generated by the representable dynamic algebras is separable and representable. Hungarian Academy of Sciences, Budapest (1980)

24. Kleene, S.C.: Representation of events in nerve nets and finite automata. In Shannon, C.E., McCarthy, J., eds.: Automata Studies. Princeton University Press, Princeton, N.J. (1956) 3–41
25. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, London (1971)
26. Kozen, D.: A completeness theorem for Kleene algebras and the algebra of regular events. Infor. and Comput. **110**(2) (1994) 366–390
27. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18**(2) (1979) 194–211
28. Kozen, D., Tiuryn, J.: Substructural logic and partial correctness. Trans. Computational Logic **4**(3) (2003) 355–378
29. Möller, B., Struth, G.: Greedy-like algorithms in Kleene algebra. In: Proc. 2nd Int. Workshop on Applications of Kleene Algebra. (2003) 173–180
30. Ehm, T., Möller, B., Struth, G.: Kleene modules. In: Proc. 2nd Int. Workshop on Applications of Kleene Algebra. (2003) 21–27
31. Kozen, D., Tiuryn, J.: On the completeness of propositional Hoare logic. Information Sciences **139**(3–4) (2001) 187–195
32. Kozen, D.: On the complexity of reasoning in Kleene algebra. Information and Computation **179** (2002) 152–162
33. Hardin, C., Kozen, D.: On the complexity of the Horn theory of REL. Technical Report 2003-1896, Computer Science Department, Cornell University (2003)
34. Hardin, C.: The Horn Theory of Relational Kleene Algebra. PhD thesis, Cornell University (2005)

From Three Ideas in TCS to Three Applications in Bioinformatics

Ming Li

School of Computer Science, University of Waterloo,
Waterloo, Ont. N2L 3G1, Canada
`mli@uwaterloo.ca`
`http://www.cs.uwaterloo.ca/~mli`

We will talk about three ideas from theoretical computer science that have actually been successfully used in real world applications in bioinformatics. From these ideas, we hope to inspire more theoretical research that are applicable to real world bioinformatics problems.

Given a gene sequence, the traditional homology search method is either too slow (dynamic programming) or not sensitive enough (BLAST). When it does return something, the results are simply some fragments of alignments instead of a complete gene match. We will explain the novel idea of optimal spaced seeds [7] that has significantly improved the sensitivity and speed of homology search. In fact, with optimized multiple spaced seeds [4], the search can be as fast as BLAST but with full sensitivity. By further integrating HMM with homology search, our homology search software returns complete gene match, with accurate intron/exon boundaries, and with high success rates [3].

The phylogenetic studies in the past have depended on individual protein sequences. However, different protein sequences often give different evolutionary histories. The genomics revolution has brought us complete genomes of many species [8]. Can we use whole genomes to obtain more robust phylogenetic trees? It turns out that the answer rests in the seemingly unlikely concept of Kolmogorov complexity [6]. We will show how to define a universal information distance [1] and its normalized version [5], and apply it to the study of evolutionary histories of species [5], chain letters [2], and many other things.

It is NP-hard to optimally align a protein sequence to a 3-D structure. This is called threading. The problem may be properly formulated as a large scale integer program. Relaxing the IP to LP, the solutions to real protein structures usually give integral hence optimal solutions. RAPTOR [9] is implemented using this idea and is a top performer in the recent CASP's.

References

1. C.H. Bennett, P. Gacs, M. Li, P. Vitanyi, and W. Zurek, Information Distance, *IEEE Trans. Inform. Theory*, 44:4(July 1998), 1407-1423. (STOC, 1993)
2. C.H. Bennett, M. Li and B. Ma, Chain letters and evolutionary histories, *Scientific American*, 288:6(June 2003) (feature article), 76-81.

3. X.F. Cui, M. Li, D. Shasha, and T. Vinar, work underway, 2006.
4. M. Li, B. Ma, D. Kisman and J. Tromp. PatternHunter II: highly sensitive and fast homology search. *J. Bioinformatics and Computational Biology*, 2:3(2004), 417–440.
5. M. Li, J. Badger, X. Chen, S. Kwong, P. Kearney, H. Zhang, An information-based sequence distance and its application to whole mitochondrial genome phylogeny, *Bioinformatics*, 17:2(2001), 149-154.
6. M. Li and P. Vitanyi, *An introduction to Kolmogorov complexity and its applications*, Springer-Verlag, 2nd Edition 1997 (xx+637 pp).
7. B. Ma, J. Tromp, M. Li, PatternHunter: Faster and more sensitive homology search. *Bioinformatics*, 18:3(2002), 440-445.
8. J.C. Wooley, Trends in computational biology. *J. Comput. Biol.* 6(1999), 459-474.
9. J. Xu, M. Li, D. Kim and Y. Xu, RAPTOR: optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1:1(2003), 95-118.

Decompositions, Partitions, and Coverings with Convex Polygons and Pseudo-triangles

O. Aichholzer[1,⋆], C. Huemer[2], S. Kappes[3], B. Speckmann[4], and C.D. Tóth[5]

[1] Institute for Software Technology, Graz University of Technology
oaich@ist.tugraz.at
[2] Departament de Matemática Aplicada II, Universitat Politécnica de Catalunya
huemer.clemens@upc.edu
[3] Department of Mathematics, TU Berlin
kappes@math.TU-Berlin.de
[4] Department of Mathematics and Computer Science, TU Eindhoven
speckman@win.tue.nl
[5] Department of Mathematics, Massachusetts Institute of Technology
toth@math.mit.edu

Abstract. We propose a novel subdivision of the plane that consists of both convex polygons and pseudo-triangles. This *pseudo-convex* decomposition is significantly sparser than either convex decompositions or pseudo-triangulations for planar point sets and simple polygons. We also introduce pseudo-convex partitions and coverings. We establish some basic properties and give combinatorial bounds on their complexity. Our upper bounds depend on new Ramsey-type results concerning disjoint empty convex k-gons in point sets.

1 Introduction

Geometric algorithms and data structures frequently use subdivisions of the input space into compact and easy to handle polygonal cells. Triangulations are among the most widely used of these tessellations. Since the running time of algorithms is often correlated with the size of the subdivision, many efficient algorithms tile the plane with generalizations of triangles such as convex polygons or pseudo-triangles which provide a sparser tessellation but retain many of the desirable properties of a triangulation. Both convex subdivisions and pseudo-triangulations have applications in areas like motion planning [7,26], collision detection [1,19], ray shooting [6,14], or visibility [22,23]. A pseudo-triangle is the "most reflex" polygon possible—it has exactly three convex vertices with internal angles less than π. Whether a chain of points is considered convex or reflex depends only on the point of view. So pseudo-triangles can be considered as natural counterparts of convex polygons.

In this paper we propose a combination of convex and pseudo-triangular subdivisions: *Pseudo-convex* decompositions. A pseudo-convex decomposition is

⋆ Research was partially supported by the FWF Joint Research Project Industrial Geometry S9205-N12.

a tiling of the plane with convex polygons and pseudo-triangles. We also introduce the related concepts of pseudo-convex partitions and coverings whose convex counterparts have been extensively studied as well. We establish some basic combinatorial properties and give quantitative bounds on the complexity of pseudo-convex decompositions, partitions, and coverings for point sets and simple polygons. Pseudo-convex decompositions are significantly sparser than convex decompositions or pseudo-triangulations.

All our bounds are combinatorial, we do in fact not know what the complexity of finding a minimum decomposition for a given input point set is. Our upper bounds depend on optimal solutions for small point configurations. Any improvement on a finite point set would lead to better bounds. We achieve optimal bounds for small configurations by proving two geometric Ramsey-type results concerning disjoint empty convex k-gons in point sets. These results extend previous work by Erdős, Hosono, and Urabe, but to the best of our knowledge our results are the first Ramsey-type answers for such questions. Small configurations of points are notoriously hard to deal with. An asymptotic lower bound for the number of order types of a set of n points in the plane is $n^{\Theta(n \log n)}$ [13]. We confirmed our conjectures regarding sets of 8 and 11 points with the help of the order type data base developed at TU Graz [2,3]. We give analytical proofs for some of our results, while others are purely based on the data base.

Organization. The next paragraphs give precise definitions for convex and pseudo-convex decompositions, partitions, and coverings and Section 2 collects some of their basic combinatorial properties. In the next subsection we state our results and compare our bounds to previous work. Pseudo-convex decompositions and partitions are significantly sparser than their convex counterparts while pseudo-convex and convex coverings have asymptotically the same complexity. We devote Section 3 to pseudo-convex decompositions and Section 4 to pseudo-convex partitions of point sets but do not discuss pseudo-convex coverings any further in this paper. Finally, Section 5 discusses pseudo-convex decompositions for the interior of simple polygons. We conclude with some open problems.

Definitions. Let S be a set of n points in general position in the plane. A *pseudo-triangle* is a planar polygon that has exactly three convex vertices with internal angles less than π, all other vertices are concave. A *pseudo-triangulation* of S is a subdivision of the convex hull of S into pseudo-triangles whose vertex set is exactly S. A vertex is called *pointed* if it has an adjacent angle greater than π. A planar straight line graph is pointed if every vertex is pointed.

The *convex decomposition number* of S, $\kappa_d(S)$, is the minimum number of faces in a subdivision of the convex hull of S into convex polygons whose vertex set is exactly S. A *pseudo-convex decomposition* of S is a partition of the convex hull of S into convex polygons and/or pseudo-triangles spanned by S. For instance every triangulation or pseudo-triangulation of S is a pseudo-convex decomposition. The *pseudo-convex decomposition number* of S, $\psi_d(S)$, is the minimum number of faces in a pseudo-convex decomposition of S. The pseudo-convex decomposition number (and equivalently the convex decomposition number) for all sets S of fixed size n is denoted by $\psi_d(n) := max_S \psi_d(S)$.

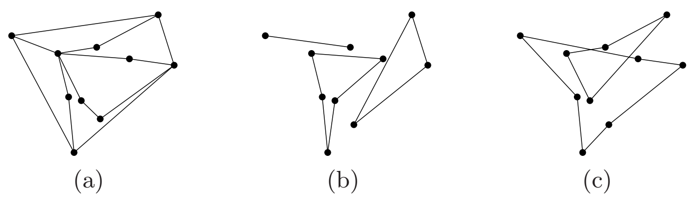


Fig. 1. A pseudo-convex decomposition (a), a pseudo-convex partition (b), and a pseudo-convex covering (c)

The *convex partition number* of S, $\kappa_p(S)$, is the minimum number of *disjoint* convex polygons spanned by S and covering all vertices of S. Similarly, the *pseudo-convex partition number* of S, $\psi_p(S)$, is the minimum number of *disjoint* convex polygons and/or pseudo-triangles spanned by S and covering all vertices of S. The pseudo-convex partition number (and equivalently the convex partition number) for all sets S of fixed size n is denoted by $\psi_p(n) := max_S \psi_p(S)$. Note that disjoint here implies empty (of points): neither a convex nor a pseudo-convex partition contains nested polygons.

The *convex cover number* of S, $\kappa_c(S)$, is the minimum number of convex polygons spanned by S and covering all points of S. Similarly, the *pseudo-convex cover number* of S, $\psi_c(S)$, is the minimum number of convex polygons and/or pseudo-triangles spanned by S and covering all points of S. The pseudo-convex cover number (and equivalently the convex cover number) for all sets S of fixed size n is denoted by $\psi_c(n) := max_S \psi_c(S)$.

1.1 Previous Work and Results

Decomposition. The convex decomposition number $\kappa_d(n)$ is bounded by

$$\frac{12}{11}n - 2 < \kappa_d(n) \leq \frac{10n - 18}{7} .$$

The lower bound was given very recently by García-López and Nicolás [11] and the upper bound was established by Neumann-Lara et al. [21]. Fevens, Meijer, and Rappaport [10] and Spillner [25] designed algorithms for computing a minimum convex decomposition for input point sets. Every minimum pseudo-triangulation of n points has exactly $n-2$ pseudo-triangles [26]. We show that the pseudo-convex decomposition number is bounded by

$$\frac{3}{5}n \leq \psi_d(n) \leq \frac{7}{10}n .$$

Furthermore, we also prove that $\psi_d(n)$ is monotonically increasing with n.

Partition. The convex partition number $\kappa_p(n)$ is bounded by

$$\left\lceil \frac{n-1}{4} \right\rceil \leq \kappa_p(n) \leq \left\lceil \frac{5n}{18} \right\rceil .$$

The lower bound was given by Urabe [27] and the upper bound was established by Hosono and Urabe [16]. Arkin et al. [4] study questions related to convex partitions and coverings by examining the reflexivity of point sets. We show that the pseudo-convex partition number $\psi_p(n)$ is bounded by

$$\left\lfloor \frac{3n}{16} \right\rfloor \leq \psi_p(n) \leq \frac{n}{4} .$$

Covering. The study of convex cover numbers is rooted in the classical work of Erdős and Szekeres [8,9] who showed that any set of n points contains a convex subset of size $\Omega(\log n)$. More recent results include the work by Urabe [27] who proved that the convex cover number $\kappa_c(n)$ is bounded by

$$\frac{n}{\log_2 n + 2} < \kappa_c(n) < \frac{2n}{\log_2 n - \log_2 e} .$$

There is an easy connection between the pseudo-convex cover number and the convex cover number, namely $\psi_c(n) \leq \kappa_c(n) \leq 3\psi_c(n)$ (all points which can be covered by a pseudo-triangle can be covered by at most three convex sets). Thus both numbers have the same asymptotic behavior, which implies

$$\psi_c(n) = \Theta\left(\frac{n}{\log n}\right) .$$

Geometric Ramsey-type Results. The upper bound construction for $\psi_d(n)$ relies on minimal pseudo-convex decomposition numbers for few points. These are, in turn, related to a combinatorial geometry problem on empty convex polygons that goes back to Erdős: For $k \geq 3$ find the smallest integer $E(k)$ such that any set S of $E(k)$ points contains the vertex set of a convex k-gon whose interior does not contain any points of S (that is, S contains an empty convex k-gon). Klein [8] showed that every set of 5 points contains an empty convex quadrilateral, that is $E(4) = 5$. Harborth [15] proved that every set of 10 points contains an empty convex pentagon, that is $E(5) = 10$. In the last decade, Urabe [27] proved that every set of 7 points can be partitioned into a triangle and a disjoint convex quadrilateral. Hosono and Urabe [16] showed that every set of 9 points contains two disjoint empty convex quadrilaterals. Very recently Gerken showed that any set that contains a convex 9-gon also contains an empty convex hexagon. Each of these results corresponds to a bound on the pseudo-convex decomposition number $\psi_d(n)$. The best upper bound we achieved depends on new results for empty convex polygons.

A typical Ramsey type problem asks for the minimum size of a system that contains at least one of two (or more) subconfigurations. We prove the following two results:

Theorem 1. *Every set of* 8 *points in general position contains either an empty convex pentagon or two disjoint empty convex quadrilaterals.*

Theorem 2. *Every set of* 11 *points in general position contains either an empty convex hexagon or an empty convex pentagon and a disjoint empty convex quadrilateral.*

Both results were established with the help of the order type data base [2,3]. In the full paper we also provide a surprisingly intuitive geometric proof of Theorem 1 that requires only a moderate number of case distinctions.

Simple Polygons. An initial step of many algorithms on simple polygons is a decomposition into simpler components [17]. Keil and Snoeyink [18] devised an algorithm for computing the minimum convex decomposition of the interior of a given simple polygon. Chazelle and Dobkin [5] studied a variant of this optimization problem allowing Steiner points, Lien and Amato [20] constructed approximately convex decompositions. Motivated by early results which we obtained during the investigations for this paper, Gerdijkov and Wolff [12] extended the work by Keil and Snoeyink to compute the minimum pseudo-convex decomposition of a simple polygon.

The minimum convex decomposition of a pseudo-triangle with n vertices may require $n-2$ triangles and the minimum pseudo-triangulation of any convex n-gon is a triangulation with $n-2$ faces. (In these extremal examples, Steiner points do not lead to a smaller convex decomposition or pseudo-triangulation.) We show that any n-gon has a pseudo-convex decomposition of size $\lceil n/2 \rceil - 1$.

Note that any quadrangulation (a decomposition into quadrilaterals) of an n-gon is a pseudo-convex decomposition, and it also has $\lceil n/2 \rceil - 1$ faces. However, not every polygon has a quadrangulation. Allowing Steiner points on the boundary of the polygon, Ramaswami, Ramos, and Toussaint [24] show that the minimum quadrangulation of every n-gon has at most $\lfloor 2n/3 \rfloor + O(1)$ faces in the worst case.

2 Basic Combinatorial Properties

Our first (trivial) observation is that $\psi_d(n) \leq \kappa_d(n)$, $\psi_p(n) \leq \kappa_p(n)$, and $\psi_c(n) \leq \kappa_c(n)$. It is well known that $\kappa_c(n) \leq \kappa_p(n) \leq \kappa_d(n)$. For pseudo-convex faces we trivially have $\psi_c(n) \leq \psi_p(n)$. $\psi_p(n) \leq \psi_d(n)$ follows from the bounds given in the previous section.

Next we observe that $\psi_d(n+1) \leq \psi_d(n) + 1$, $\psi_p(n+1) \leq \psi_p(n) + 1$, and $\psi_c(n+1) \leq \psi_c(n) + 1$. This follows by induction when inserting the points in x-sorted order. For covering and partitioning the last inserted vertex is a singleton, for decomposing it forms a corner of a pseudo-triangle similar to the last step in a Henneberg construction.

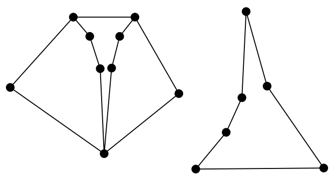


Fig. 2. Sets with non-monotone behavior

The following lemma establishes an interesting connection between the convex partition number and the pseudo-convex decomposition number.

Lemma 1. *For any point set S we have $\psi_d(S) \leq 3\kappa_p(S) - 2$ and thus $\psi_d(n) \leq 3\kappa_p(n) - 2$.*

Table 1. Bounds on the pseudo-convex cover number $\psi_c(n)$, partition number $\psi_p(n)$, and decomposition number $\psi_d(n)$ for small point sets

n	3	4	5	6	7	8	9	10	11	12	13	14	15
$\psi_c(n)$	1	1	2	2	2	2	2	3	3	3	3	3	3
$\psi_p(n)$	1	1	2	2	2	2	3	3	3	3	3..4	3..4	4
$\psi_d(n)$	1	2	2	3	4	4	5	6	6	7	8	8..9	8..9

The pseudo-convex decomposition, partition, and covering numbers for a particular point set S are not necessarily monotone. Consider the examples in Figure 2. On the left, a set S with 9 points and $\psi_d(S) = 3$. Removing the bottom most point of S results in a set S' with 8 points and $\psi_d(S') = 4$. On the right, a set S with 6 points and $\psi_c(S) = \psi_p(S) = 1$. Removing the top-most point of S results in a set S' with 5 points and $\psi_c(S') = \psi_p(S') = 2$. Table 1 shows the exact values of $\psi_c(n)$, $\psi_p(n)$, and $\psi_d(n)$ for small sets of points.

3 Pseudo-convex Decompositions

We first give a formula for the number of faces in a pseudo-convex decomposition:

Lemma 2. *Let S be a set of n points in general position. Let P be a pseudo-convex decomposition of S, n_k the number of convex k-gons in P, and p the number of pointed vertices. Then the number of faces of P is*

$$|P| = 2n - p - 2 - \sum_{k=4}^{n} n_k(k-3)$$

Corollary 3. *The number of faces in a pointed pseudo-convex decomposition is*

$$|P| = n - 2 - \sum_{k=4}^{n} n_k(k-3)$$

Although the pseudo-convex decomposition number for a particular point set S might not be monotone (recall Figure 2), $\psi_d(n)$ nevertheless increases monotonically with n.

Theorem 4. *The pseudo-convex decomposition number increases monotonically with the number of points.*

3.1 Small Point Sets

In this section we give tight upper and lower bounds on $\psi_d(n)$ for sets of up to 13 points. Recall that $\psi_d(n+1) \leq \psi_d(n) + 1$ and (by Theorem 4) $\psi_d(n) \leq \psi_d(n+1)$. Obviously $\psi_d(3) = 1$. If four points do not lie in convex position (see Fig. 3(a)) then any decomposition needs at least two faces and hence $\psi_d(4) = 2$

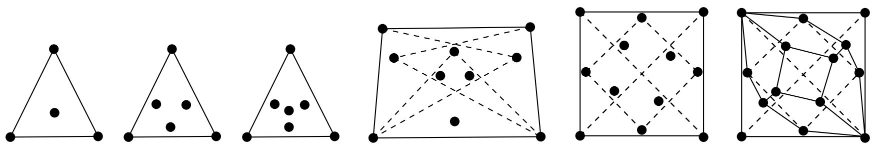


Fig. 3. (a)-(e) Lower bound examples, (f) every minimum decomposition is non-pointed

and $\psi_d(5) \geq 2$. Every set of 5 points contains an empty convex quadrilateral [8]. Pseudo-triangulating in a pointed way around this quadrilateral yields $\psi_d(5) = 2$ by Corollary 3.

$\psi_d(5) = 2$ implies $\psi_d(6) \leq 3$. Figure 3(b) shows a configuration S of 6 points such that every pseudo-convex decomposition of S has at least 3 faces. S does not span any empty convex k-gon for $k > 4$. Any empty convex quadrilateral spanned by S necessarily uses all three inner points, so any partition of S can contain at most one convex quadrilateral which implies $\psi_d(6) = 6-2-(4-3) = 3$ for pointed pseudo-decompositions which are optimal in this case.

$\psi_d(6) = 3$ implies $\psi_d(7) \leq 4$. Figure 3(c) shows a configuration S of 7 points such that every pseudo-convex decomposition of S has at least 4 faces. The argument is similar to the one for the example with 6 points. Again, S does not span any empty convex k-gon for $k > 4$. Any pointed decomposition contains at most one convex quadrilateral, because every convex quadrilateral contains the point in the center. With every additional quadrilateral, we also add at least one non-pointed vertex, so a non-pointed decomposition cannot contain less faces than a pointed one. Therefore, $\psi_d(7) = 7 - 2 - (4 - 3) = 4$.

$\psi_d(7) = 4$ implies $\psi_d(8) \geq 4$. Theorem 1 together with Corollary 3 implies $\psi_d(8) \leq 8 - 2 - 2 = 4$. We construct this decomposition by pseudo-triangulating in a pointed way around the convex polygon(s) guaranteed by Theorem 1.

Every set of 10 points contains an empty pentagon [15] and so Corollary 3 implies $\psi_d(10) \leq 10 - 2 - (5 - 3) = 6$. Figure 3(d) (which is a close relative of a construction in [16]) shows a configuration S of 10 points such that every pseudo-convex decomposition of S has at least 6 faces. First note that S does not span an empty convex pentagon and a disjoint empty convex quadrilateral. Furthermore, every empty convex pentagon spanned by S necessarily contains the three points in the upper center, so any partition of S can contain at most one convex pentagon. If we start our decomposition with a pentagon, then we can not add a quadrilateral without creating at least one non-pointed vertex. Therefore, any non-pointed decomposition cannot save any faces compared to the pointed one which implies $\psi_d(10) = 10 - 2 - (5 - 3) = 6$.

$\psi_d(10) = 6$ implies that $\psi_d(9) \geq 5$. Since every set of 9 points contains two disjoint empty convex quadrilaterals [16], we have (with Corollary 3) $\psi_d(9) \leq 9-2-2*(4-3) = 5$. $\psi_d(10) = 6$ also implies $\psi_d(11) \geq 6$. Theorem 2 together with Corollary 3 yields $\psi_d(11) \leq 11 - 2 - 3 = 6$. We construct this decomposition by pseudo-triangulating in a pointed way around the convex polygon(s) guaranteed by Theorem 2.

$\psi_d(11) = 6$ implies $\psi_d(12) \leq 7$. Figure 3(e) shows a configuration S of 12 points such that every pseudo-convex decomposition of S has at least 7 faces. The largest empty convex set in this configuration is a hexagon. Every empty convex pentagon or hexagon contains at least three of the four inner points and thus separates the other points, so that no disjoint convex quadrilateral can be found. The coordinates of this point set are: $(0,0)$, $(0,20)$, $(20,20)$, $(20,0)$, $(1,10)$, $(10,19)$, $(19,10)$, $(10,1)$, $(5,7)$, $(7,15)$, $(15,13)$, $(13,5)$.

$\psi_d(12) = 7$ implies $\psi_d(13) \leq 8$. The point set with the following coordinates requires 8 faces for every pseudo-convex decomposition: $(65535, 65535)$, $(0,0)$, $(29293, 36890)$, $(15166, 26472)$, $(27461, 37283)$, $(32929, 42217)$, $(29439, 42711)$, $(27746, 42587)$, $(27491, 42925)$, $(32135, 45720)$, $(29447, 45175)$, $(31736, 48764)$, $(19257, 42830)$.

3.2 Upper Bound

Our upper bound construction is based on exact pseudo-convex decomposition numbers for small point sets. Assume that we are given a set S with n points and that we know the value of $\psi_d(k)$ for some $k < n$. We choose a point p on the convex hull of S. Now we partition the plane by half-lines emanating from p into $\lceil (n-1)/(k-1) \rceil$ wedges such that every wedge contains at most $k-1$ points of $S \setminus \{p\}$. Let a *petal* be the convex hull of points in a wedge together with p. We have a total of $\lceil (n-1)/(k-1) \rceil$ petals, each of which can be decomposed into at most $\psi_d(k)$ faces. Two adjacent petals can be combined with a pseudo-triangle into one larger convex set. We combine inductively adjacent convex sets (all including p) until we obtain the convex hull of S. We have proved an upper bound of

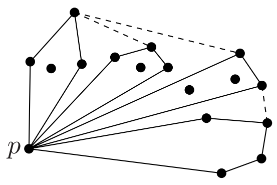


Fig. 4. Petals of size 5

$$\psi_d(n) \leq \left\lceil \frac{n-1}{k-1} \right\rceil \psi_d(k) + \left\lceil \frac{n-1}{k-1} \right\rceil - 1 \leq \frac{\psi_d(k)+1}{k-1} n \ . \tag{1}$$

The best currently known upper bound can be achieved by evaluating Inequality (1) for $k = 11$ and $\psi_d(11) = 6$. We obtain

$$\psi_d(n) \leq \frac{\psi_d(11)+1}{11-1} n = \frac{6+1}{10} n = \frac{7n}{10} \ .$$

Furthermore, the left inequality of (1) implies $\psi_d(15) \leq 9$ for $k = 8$.

3.3 Lower Bound

We present a lower bound construction of $5k$ points for every odd $k \geq 3$ such that any pseudo-convex decomposition consists of at least $3k - 1$ faces (see Fig. 5). The details of the construction can be found in the full paper. It implies

$$\psi_d(n) \geq \frac{3n}{5} - 1.$$

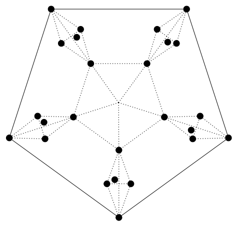


Fig. 5. Lower bound example for $k = 5$

4 Pseudo-convex Partitions

An upper bound of $\psi_p(n) \leq n/4$ can be easily established: Any four points form either a pseudo-triangle or a convex quadrilateral and grouping them in x-sorted order guarantees disjointness. It is possible that optimal bounds on small point sets improve the upper bound of $n/4$. For example, we do not know the exact value of $\psi_p(13)$, we know only that $\psi_p(13) \in \{3, 4\}$ (c.f., Table 1). $\psi_p(13) = 3$ would imply $\psi_p(n) \leq 3n/13$ by partitioning x-sorted groups of 13 points independently.

4.1 Lower Bound

Lemma 3. $\psi_p(n) \geq \lfloor \frac{3n}{16} \rfloor$.

Proof. We consider a set S of $n = 4k$ points (see Fig. 6). S consists of k groups of 4 points, a_i, b_i, c_i, and d_i. First we show that if c_i is a reflex vertex of a pseudo-triangle P, then a_i and b_i must be the corners of P: this is the case since c_i lies in the convex hull of the corners of P, and there is a halfplane for a_i (b_i) whose boundary line passes through c_i and whose intersection with P is a_i (b_i).

Let $W \subset S$ denote a subset of $3k$ points $\{a_i, b_i, c_i : i = 1, 2, \ldots, k\}$. Consider a polygon P from a pseudo-convex partition of S. We show next that P is incident to at most 4 points of W. This implies immediately that any pseudo-convex partition of these $n = 4k$ points consists of at least $3k/4 = 3n/16$ polygons. Suppose, by contradiction, that P is incident to more than 4 points of W.

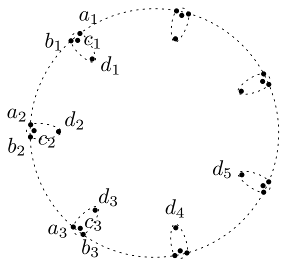


Fig. 6. $k = 7$

First suppose that P is convex, that is, P contains a convex pentagon Q with all vertices in W. Since each group contains only three points of W, Q must have corners in at least two groups. Q can contain at most two points from each group, because the triangle $a_i b_i c_i$ cannot be completed to a convex pentagon in S. Therefore, Q must have corners in at least three groups, and it contains a triangle T with corners of W from three different groups. We show that T (and also P) contains a point d_i in its interior, which is a contradiction. If T has a corner in W in group j, then T contains the point d_j in its interior unless both other corners must be either in groups $[j+1, j+\lfloor k/2 \rfloor]$ or groups $[j + \lceil k/2 \rceil, j + k - 1]$. There are no three groups whose indices satisfy these constraints for all three corners, and so T must contain a point d_i in its interior.

If P is a pseudo-triangle with at least five vertices from W, then it must have two reflex vertices from W. Since the convex hull vertices can only be corners of P, two reflex vertices are c_i and c_j, $i \neq j$. We have seen that if P contains c_i and c_j, then it also contains a_i, b_i and a_j, b_j, and so it must have four corners: A contradiction. □

5 Pseudo-convex Decompositions of the Interior of a Simple Polygon

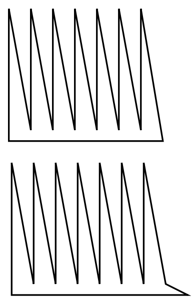
Theorem 5. *Every simple polygon with $n \geq 3$ vertices has a decomposition into at most $\lceil \frac{n-2}{2} \rceil$ convex or pseudo-triangular faces, and this is the best possible bound.*

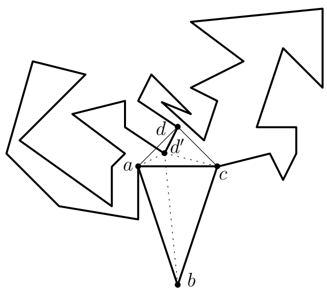
Proof. The lower bound is attained by the comb polygons (Fig. 7 (a)). We prove the upper bound by induction on $n \in \mathbb{N}$. The theorem is obvious for $n = 3, 4$. Consider a simple polygon P_n with $n \geq 5$ vertices. Triangulate P_n and let T_n denote the dual graph of the triangulation. Every node of T_n corresponds to a triangle, and every edge of T_n corresponds to a diagonal in the triangulation. T_n is a tree with maximal degree three and with $n-2$ nodes.

If n is odd then we delete a triangle t corresponding to a leaf node in T_n. By induction, $P_n - t$ can be decomposed into $\frac{n-3}{2}$ faces. Therefore P_n decomposes into $\frac{n-3}{2} + 1 = \lceil \frac{n-2}{2} \rceil$ faces. Assume that n is even, and so $\lceil \frac{n-2}{2} \rceil = \frac{n}{2} - 1$. The triangulation consists of an even number of triangles. If a diagonal decomposes P_n into two even polygons, then induction completes the proof. Hence we assume that every diagonal decomposes P_n into two odd polygons.

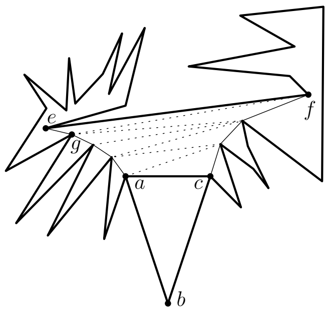
Let the triangle abc correspond to a leaf in T_n such that ac is a diagonal of P_n. We show that no diagonal of P_n is incident to b. Suppose, by contradiction, that ad is a diagonal of P_n. Then $abcd$ is a convex polygon, let d' be the vertex of P_n in $acd \setminus \{a, c\}$ closest to the line ac. Note that bd' is a diagonal of P_n, and at least one of ad' and cd' is also a diagonal (since $n \geq 5$). If bd' decomposes P_n into odd polygons, then either ad' or cd' decomposes it into two (non-empty) even polygons. We conclude that b sees the interior of an edge ef of P_n.

Consider the pseudo-triangle $\mathrm{pt}(b, e, f)$ (three corners uniquely define a pseudo-triangle in a simple polygon). If $P_n = \mathrm{pt}(b, e, f)$, then P_n is a pseudo-triangle, and our proof is complete. Each of the components of $P_n - \mathrm{pt}(b, e, f)$ is an odd polygon. Every such component is adjacent to a unique edge of the geodesic $\mathrm{geo}(a, e)$ or


(a) Comb polygon for n odd and for n even.



(b) If b is incident to a diagonal bd, then there is a vertex d' such that bd' and at least one of ad' or cd' are also diagonals.



(c) If b sees the edge ef then we can form the pseudo-triangle $\mathrm{pt}(b, e, f)$.

Fig. 7. Lower bound (a). An example 24-gon. (b)-(c).

geo(c, f). If pt(b, e, f) has k vertices, then it has $k-3$ edges along these geodesics (all edges except ab, bc, and ef). We show that there is one edge along the geodesics geo(a, e) and geo(c, f) that is not adjacent to any component of $P_n - \text{pt}(b, e, f)$: Consider the dual graph of an arbitrary triangulation of pt(b, e, f). It is a tree where one leaf node corresponds to abc and another leaf corresponds to efg for some vertex g. Assume w.l.o.g. that eg is a side and fg is a diagonal in pt(b, e, f). If eg were adjacent to an odd component of $P_n - \text{pt}(b, e, f)$, then fg would partition P_n into two even polygons. Therefore pt(b, e, f) with k vertices is adjacent to at most $k-4$ components of $P_n - \text{pt}(b, e, f)$.

Let n_i denote the number of vertices of the components of $P_n - \text{pt}(b, e, f)$ for $i = 1, 2, \ldots, k-4$. We have $k + \sum_{i=1}^{k-4}(n_i - 2) = n$. By induction, every odd component with n_i vertices can be decomposed into $(n_i - 1)/2$ faces. Together with pt(b, e, f), the polygon P_n can be decomposed into

$$1 + \sum_{i=1}^{k-4} \frac{n_i - 1}{2} \leq 1 + \frac{1}{2}\left(\sum_{i=1}^{k-4} n_i - 2\right) + \frac{k-4}{2} = \frac{n}{2} - 1$$

faces, as required. □

6 Conclusions and Open Problems

We proposed pseudo-convex decompositions, partitions, and coverings. We established some of their basic properties and gave combinatorial bounds on their complexity. Our upper bounds depend on new Ramsey-type results concerning disjoint empty convex k-gons in the plane. We (obviously) would like to know what the exact bounds on $\psi_d(n)$ and $\psi_p(n)$ are and if the exact bound for $\psi_d(n)$ can be realized with a pointed decomposition. It would also be interesting to determine the complexity of computing a minimum pseudo-convex decomposition or covering for a given point set.

Acknowledgements. The first two authors want to thank Ferran Hurtado and Hannes Krasser for valuable discussions on the presented subject.

References

1. P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection. *International Journal of Robotics Research*, 21:179–197, 2002.
2. O. Aichholzer, F. Aurenhammer, and H. Krasser. Enumerating order types for small point sets with applications. *Order*, 19:265–281, 2002.
3. O. Aichholzer and H. Krasser. Abstract order type extensions and new results on the rectilinear crossing number. In *Proc. 21st Symposium on Computational Geometry*, pages 91–98, 2005.
4. E. M. Arkin, S. P. Fekete, F. Hurtado, J. S. B. Mitchell, M. Noy, V. Sacristán, and S. Sethia. On the reflexivity of point sets. In *Discrete and Comput. Geometry: The Goodman-Pollack Festschrift*, volume 25, pages 139–156. Springer-Verlag, 2003.

5. B. Chazelle and D. Dobkin. Optimal convex decompositions. In *Computational Geometry* (G. T. Toussaint, ed.), pages 63–133. North-Holland, 1985.
6. B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12:54–68, 1994.
7. M. de Berg, J. Matoušek, and O. Schwarzkopf. Piecewise linear paths among convex obstacles. *Discrete and Computational Geometry* 14(1):9–29, 1995.
8. P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.
9. P. Erdős and G. Szekeres. On some extremum problem in geometry. *Annales Univ. Sci. Budapest*, 3-4:53–62, 1960.
10. T. Fevens, H, Meijer, and D. Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109:95–107, 2001.
11. J. García-López and M. Nicolás. Planar point sets with large minimum convex partitions. In *Abstr. 22nd European Workshop on Computational Geometry*, pages 51–54, 2006.
12. S. Gerdjikov and A. Wolff. Pseudo-convex decomposition of simple polygons. In *Abstr. 22nd European Workshop on Computational Geometry*, pages 13–16, 2006.
13. J. Goodman and R. Pollack. Allowable sequences and order types in discrete and computational geometry. In *New Trends in Discrete and Computational Geometry*, Springer-Verlag, New York, pages 103–134, 1993.
14. M. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths in planar subdivision via balanced geodesic triangulations. *J. of Algorithms*, 23:51–73, 1997.
15. H. Harborth. Konvexe Fünfecke in ebenen Punktmengen. *Elemente der Mathematik*, 33(5):116–118, 1978.
16. K. Hosono and M. Urabe. On the number of disjoint convex quadrilaterals for a planar point set. *Comput. Geom.: Theory and Applications*, 20:97–104, 2001.
17. J. M. Keil. Decomposing a polygon into simpler components, *SIAM Journal on Computing*, 14:799-817, 1985.
18. J. M. Keil and J. Snoeyink. On the time bound for convex decomposition of simple polygons. *Intern. J. Comput. Geom. Appl.,* 12:181–192, 2002.
19. D. Kirkpatrick and B. Speckmann. Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons. In *Proc. 18th Symposium on Computational Geometry*, pages 179–188, 2002.
20. J-M. Lien and N. M. Amato. Approximate convex decomposition. In *Proc. 20th Symposium on Computational Geometry*, pages 17-26, 2004.
21. V. Neumann-Lara, E. Rivera-Campo, and J. Urrutia. A note on convex decompositions of a set of points in the plane. *Graphs and Combinatorics*, 20(2):223–231, 2004.
22. J. O'Rourke. Visibility. In *Handbook of Discrete and Computational Geometry* (2nd ed.), CRC Press, Boca Raton, pages 643–664, 1997.
23. M. Pocchiola and G. Vegter. Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete and Computational Geometry*, 16:419–453, 1996.
24. S. Ramaswami, P. A. Ramos, and G. T. Toussaint. Converting triangulations to quadrangulations. *Comput. Geom.: Theory and Applications*, 9:257–276, 1998.
25. A. Spillner. Optimal convex partitions of point sets with few inner points. In *Proc. 17th Canadian Conference on Computational Geometry*, pages 39–42, 2005.
26. I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proc. 41st Symp. Found. Comp. Science*, pages 443–453, 2000.
27. M. Urabe. On a partition into convex polygons. *Discrete Applied Mathematics*, 64:179–191, 1996.

Approximate Shortest Path Queries on Weighted Polyhedral Surfaces*

Lyudmil Aleksandrov[1], Hristo N. Djidjev[2], Hua Guo[3], Anil Maheshwari[3], Doron Nussbaum[3], and Jörg-Rüdiger Sack[3]

[1] Bulgarian Academy of Sciences, Sofia, Bulgaria
[2] Los Alamos National Laboratory, Los Alamos, USA
[3] School of Computer Science, Carleton University, Ottawa, Canada

Abstract. We consider the classical geometric problem of determining shortest paths between pairs of points lying on a weighted polyhedral surface P consisting of n triangular faces. We present query algorithms that compute approximate distances and/or approximate (weighted) shortest paths. Our algorithm takes as input an approximation parameter $\varepsilon \in (0, 1)$ and a query time parameter $\mathfrak{q}$ and builds a data structure which is then used for answering ϵ-approximate distance queries in $O(\mathfrak{q})$ time. This algorithm is source point independent and improves significantly on the best previous solution. For the case where one of the query points is fixed we build a data structure that can answer ϵ-approximate distance queries to any query point in P in $O(\log \frac{1}{\varepsilon})$ time. This is an improvement upon the previously known solution for the Euclidean fixed source query problem. Our algorithm also generalizes the setting from previously studied unweighted polyhedral to weighted polyhedral surfaces of arbitrary genus. Our solutions are based on a novel graph separator algorithm introduced here which extends and generalizes previously known separator algorithms.

1 Introduction

Problem Definition. Shortest path problems are among the fundamental problems studied in computational geometry, network optimization, graph algorithms, geographical information systems, and calculus of variations. In this paper we study paths that stay on the surface of a connected polyhedral surface[1] P of genus g in the 3-dimensional Euclidean space consisting of n positively weighted triangular faces. The cost of a path lying inside a face is its Euclidean length multiplied by the weight of the face. The cost of a path on P is the sum of the costs of the sub-paths within each face traversed. For a pair of points on P the path of least cost between them is called *shortest path*. The cost of the shortest path is called *distance* between its end-points.

* Research supported by NSERC, SUN Microsystems, and a P.E.O. Scholar Award. The first and the second author are Adjunct Professors at Carleton University.

[1] Surface P can be any polyhedral 2-manifold without assumed additional geometrical/topological properties like convexity, being a terrain, absence of holes, etc.

Throughout the paper ε is a user-specified accuracy parameter, i.e., a fixed real number in $(0, 1)$. A path whose cost divided by the cost of the shortest path is in $(1-\varepsilon, 1+\varepsilon)$ is called *ε-approximate (or simply approximate) shortest path*. The cost of an approximate path is called *approximate distance*. The approximate distance (and/or shortest path) query problem is: Preprocess the surface P so that for a pair of query points a and b approximate distance and/or an approximate shortest path between a and b can be answered efficiently. We consider the following standard two variations of these problems: The *Single Source Query* (SSQ) problem in which one of the query points is fixed and the other one is any point on P and the *All Pairs Query* (APQ) problem in which the query consists of two arbitrary points in P. To place our work in the context of the literature we next state some relevant results. (We assume that the faces containing the query points are already known.)

Previous Work. A lot of research work has been carried out to solve shortest path problems in planar graphs. Here we are interested in the geometric setting, where complexities tend to be much higher than for planar graphs in particular, for geometric weighted shortest paths computations (see [6] for references). If P is a convex surface, then a convex set Q of size $O(\frac{1}{\varepsilon^{3/2}})$ exists, such that $P \subset Q$ and the Hausdorff distance between P and Q is $\varepsilon \cdot \text{diameter}(P)$. This was used in the algorithm of [2] to compute an ε-approximate shortest path in $O(n \log \frac{1}{\varepsilon} + \frac{1}{\varepsilon^3})$ time. In [13], this algorithm was extended to answer APQs in $O(\log n/\varepsilon^{1.5} + 1/\varepsilon^3)$ time. Chazelle et al. [7] very recently presented a sublinear randomized algorithm for solving APQs on convex polyhedral surfaces. A preliminary result on APQ for weighted polyhedra has been announced in [5]. We summarize other results relevant to this paper in Table 1.

New Results. The main of results of this paper are

1. New graph partitioning algorithms (Section 2) which partition embedded graphs of genus g with weights and costs assigned to their vertices into components of specified weight so that their boundaries have small cost. These results extend and/or improve upon results in [3,4,10,11,12]. For the design of the APQ algorithm we employed this new partitioning algorithm.

2. We present a novel algorithm (Section 5) for solving the approximate APQ problem in weighted polyhedral surfaces P of arbitrary genus g. The algorithm takes as input a query time parameter $\mathfrak{q}$ within a certain range and builds a data structure $\text{APQ}(P, \varepsilon; \mathfrak{q})$ to answer approximate distance and/or shortest path queries between arbitrary points in P in $O(\mathfrak{q})$ time.

3. We derive an algorithm (Section 4) for solving the approximate SSQ problem in weighted polyhedral surfaces of arbitrary genus. See the corresponding entry in Table 1. This result improves upon the previous result in [14] in three ways. First, it works for weighted surfaces of arbitrary genus. Second, the preprocessing time is reduced by a factor of n. Third, the size of the data structure and the preprocessing time are reduced by at least a factor of $\sqrt{\varepsilon}$.

Our Approach. At the core of our approach is the discretization method (Section 3) and the Single Source Shortest Path (SSSP) algorithm developed in [6].

Table 1. We assume for our algorithms that the face containing the query point is known. The time complexity of our preprocessing algorithms inherits the dependency on geometric parameters from [6]. In [8] d is an adjustable parameter with $1 < d \leq n$.

Reference Surface Type	Preprocessing Time	Query Time	Space	Source
[1] convex unweighted	$O(n^6 m^{1+\delta})$	$O\left(\frac{\sqrt{n}}{m^{1/4}} \log n\right)$	$O(n^6 m^{1+\delta})$	APQ
[9] nonconvex unweighted	$O(n^{11})$	$O(\log n)$	$O(n^{12})$	APQ
[8] nonconvex unweighted	$O(n^2)$	$O(d\log_d n)$	$O(n\log_d n)$	SSQ
[14] nonconvex unweighted	$O\left(n^2 \log n + \frac{n}{\epsilon} \log \frac{1}{\epsilon} \log \frac{n}{\epsilon}\right)$	$O\left(\log \frac{n}{\epsilon}\right)$	$O\left(\frac{n}{\epsilon} \log \frac{1}{\epsilon}\right)$	SSQ
Here nonconvex weighted	$\mathbf{O\left(\frac{n}{\sqrt{\epsilon}} \log \frac{1}{\epsilon} \log \frac{n}{\epsilon}\right)}$	$\mathbf{O\left(\log \frac{1}{\epsilon}\right)}$	$\mathbf{O\left(\frac{n}{\sqrt{\epsilon}} \log \frac{1}{\epsilon}\right)}$	**SSQ**
Here nonconvex weighted	$\mathbf{O\left(\frac{(g+1)n^2}{\varepsilon^{3/2}\mathfrak{q}} \log \frac{n}{\epsilon} \log^4 \frac{1}{\epsilon}\right)}$	$\mathbf{O}(\mathfrak{q})$	$\mathbf{O\left(\frac{(g+1)n^2}{\epsilon^{3/2}\mathfrak{q}} \log^4 \frac{1}{\epsilon}\right)}$	**APQ**

The discretization method constructs a graph G_ε, called approximation graph, by inserting a set of Steiner points inside the faces of P. The edges of G_ε connect nodes incident to neighbor faces and have cost equal to the cost of the shortest "local" paths between their endpoints. Next, a highly efficient SSSP algorithm is employed for finding approximate distances from a fixed vertex of G_ε to all other vertices G_ε. Here we use a modification of the graph G_ε, which has some additional edges and the edge cost is defined in slightly different way. Our algorithm for solving the SSQ problem builds a data structure SSQ$(P, \varepsilon; a)$ consisting of a SSSP tree in G_ε rooted at the point a plus $O(n)$ local data structures related to the faces of P. Our algorithm for solving the APQ problem builds a data structure APQ$(P, \varepsilon; \mathfrak{q})$ consisting of a collection of SSQ data structures. The choice of the SSQ data structures inserted into APQ$(P, \varepsilon; \mathfrak{q})$ depends on a balanced decomposition of the surface P in terms of the number of nodes of G_ε incident to different parts.

2 Partitioning Embedded Graphs with Weights and Costs

In this section we present two new results on partitioning of embedded graphs with weights and costs assigned to vertices. We consider connected graphs $G = (V, E)$ that are 2-cell embedded onto an orientable surface of genus g where positive weights and costs are assigned to the vertices of G. For a subgraph G' of G, we denote by $w(G')$ and $c(G')$ the sum of the weights and sum of the costs of the vertices in G', respectively. Let t be a real number in $(0, 1)$. A set of vertices S of G is called a *t-separator* if its removal from G leaves no component of weight exceeding $tw(G)$. We denote the sum of the squares of the costs of the vertices of G by $\sigma(G)$, i.e. $\sigma(G) = \sum_{v \in V}(c(v))^2$. The lemma below follows directly from the method presented in [3] and using this we show how "low-cost" t-separators can be constructed.

Lemma 1. *Let K be an embedded, triangulated graph of genus γ with non-negative vertex weights. Let T be a spanning tree of K. There exists a t-separator C of K that satisfies:* **(a)** *The separator C consists of at most $4(\gamma + 1/t)$ fundamental cycles*[2]. **(b)** *Any component of $K \setminus C$ can be adjacent to at most $2(\gamma+1)$ cycles in C. Such a separator C can be constructed in $O(|K| \log |K|)$ time.*

Theorem 1. *Let G be an embedded graph of genus g with weights and costs assigned to its vertices. For any $t \in (0, 1)$ there exists a t-separator S whose cost is at most $4\sqrt{2(g + 1/t)\sigma(G)}$; S can be constructed in $O(|G| \log |G|)$ time.*

Proof: (Sketch) The theorem is proved by constructing a t-separator S whose cost is as required. S is constructed in two phases. In the first phase we "slice" the graph into subgraphs with "short" (in terms of cost) spanning trees using a single source shortest path (SSSP) tree T rooted at a vertex ρ. For any real $x \geq 0$ we define a set of vertices $L(x)$ called *level* as follows. A vertex v is in $L(x)$ if its distance to ρ is at least x and the distance of its predecessor in T to ρ is less than x. Let $h = \frac{1}{2\sqrt{2}}\sqrt{\frac{\sigma(G)}{(g+1/t)}}$. We apply the method described in [4] and compute a set of levels $\mathcal{L}_h$, so that their removal partitions G into components with SSSP trees of radius not exceeding $2h$. The vertices in the levels $\mathcal{L}_h$ are inserted into S; their total cost does not exceed $\sigma(G)/h$.

In the second phase, we use Lemma 1 to obtain a t-separator. Each "heavy" component K, i.e. $w(K) > tw(G)$, of the graph $G \setminus \mathcal{L}_h$ is further partitioned by fundamental cycles as stated in Lemma 1 with a parameter $t_K = tw(G)/w(K)$. The resulting separator $S(K)$ is inserted in S. By the construction in Phase I and by Lemma 1 the cost of the separator $S(K)$ is bounded by $c(S(K)) \leq 8(\gamma(K) + 1/t_K)h$, where $\gamma(K)$ is the genus of the subsurface containing K. From this inequality the stated bounds follow. □

Next, we extend the approach applied in the above construction to obtain "low-cost" t-separators, which partition the graph into components with "small-cost" boundaries. Any t-separator S naturally defines a partitioning of the vertices of G into sets inducing the connected components of $G \setminus S$ and S itself. Let $V_1, \ldots, V_k, S$ be the partitioning defined by a t-separator S. Note that a vertex in a set V_i for some $1 \leq i \leq k$ can be adjacent to vertices in $V_i \cup S$ only. The subset of vertices in S that are adjacent to vertices in V_i is called *boundary* of V_i (or of the component induced by V_i) and is denoted by ∂V_i. A partitioning $V_1, \ldots, V_k, S$ defined by S is called *B-regular* (or simply *regular*), where B is a real number, if the costs $c(\partial V_i)$ for $i = 1, \ldots, k$ are bounded by B.

[2] A fundamental cycle is a cycle consisting of a single non-tree edge (v_1, v_2) plus the two paths in T from v_1 and v_2 to their lowest common ancestor.

Theorem 2. *Let G be an embedded graph of genus g with maximum degree three and with weights and costs assigned to its vertices. For any $t \in (0,1)$ there exists a t-separator S, that defines a $2B$-regular partitioning of G with $B = \sqrt{(g+1)t\sigma(G)}$, whose cost is $O\left(\sqrt{(g+1)\sigma(G)/t}\right)$. Such a separator can be constructed in $O(|G|\log|G|)$ time.*

Proof: (Sketch) We set $6h = B/(g+1) = \sqrt{t\sigma(G)/(g+1)}$ and apply Phase I as described in the proof of Theorem 1 above. Thus we compute a set of levels $\mathcal{L}_h$, whose removal partitions the graph into components, whose spanning trees have radii (in terms of cost) are bounded by $2h$ and the cost of vertices in $\mathcal{L}_h$ is $c(\mathcal{L}_h) \leq \sigma(G)/h$. Then we apply Phase II and obtain a set of fundamental cycles C_1, such that the set of vertices in $S_1 = \mathcal{L}_h \cup C_1$ is a t-separator for G and their cost is $c(C_1) \leq 8h(g + 1/t)$; but they may not induce a $2B$-regular partitioning since there might be components in $G \setminus S_1$ with boundaries whose cost exceeds $2B$.

Let K be a component of $G \setminus S_1$, such that $c(\partial K) > 2B$. We consider the subgraph of G induced by the set of vertices $V(K) \cup \partial K$ and denote it by $\tilde{K}$. Let the genus of the subsurface containing $\tilde{K}$ be $\gamma(K)$. We assign new weights $w_1(v)$ and costs $c_1(v)$ to the vertices of $\tilde{K}$ as follows. We denote by $\partial' K$ the set of vertices in ∂K that belong to $\mathcal{L}_h$, i.e. $\partial' K = \partial K \cap \mathcal{L}_h$. The new cost of the vertices in ∂K is set to zero and the new cost of vertices in K equals to its original cost, i.e. for $v \in K$ we have $c_1(v) = c(v)$ and for $v \in \partial K$, $c_1(v) = 0$. The new weight of a vertex v in K is the sum of the costs of the vertices in $\partial' K$ that are adjacent to v. The weights of the vertices in ∂K and the vertices in K not adjacent to $\partial' K$ are set to zero. By this definition and since the maximum degree of G is three we have $w_1(\tilde{K}) \leq 3c(\partial' K)$. Then we set $t_K = (2/3)B/w_1(\tilde{K})$ and compute a t_K-separator of $\tilde{K}$ using Lemma 1. We denote this separator by $C(K)$. It can be shown that the cost of this separator is $c(C(K)) \leq 8h\gamma(K) + 6c(\partial K \cap \mathcal{L}_h)/(g+1)$. Now for any component K_1 of $\tilde{K} \setminus C(K)$ we have $c(\partial K_1) \leq 2B$.

Define separator S to be the union of S_1 and the separators $C(K)$ computed for all components K with $c(\partial K) > 2B$. Clearly, S is a t-separator that induces a $2B$-regular partitioning of G. The cost of S can be estimated as $c(S) \leq c(S_1) + \sum_{c(\partial K)>2B} c(C(K)) \leq \sigma(G)/h + 8h(g + 1/t) + \sum_{c(\partial K)>2B} 8h(\gamma(K) + 9c(\partial K \cap \mathcal{L}_h))/2B) \leq \sigma(G)/h + 8h(2g+1)/t + 6c(\mathcal{L}_h)/(g+1) < 45\sqrt{(g+1)\sigma(G)/t}$. □

3 Approximating Shortest Paths

First we adapt the discretization scheme presented in [6] and establish properties which are beneficial for answering shortest path queries as we will show. Let P be a polyhedral surface in the 3-dimensional Euclidean space consisting of n triangular faces $f_1, \ldots, f_n$. Each face f_i has an associated positive weight w_i, representing the cost of traveling a unit Euclidean distance inside it. The weight of an edge is the minimum of the weights of the triangles incident to that edge. The cost of a path π in P is defined as $\|\pi\| = \sum_{i=1}^{n} w_i|\pi_i|$, where $|\pi_i|$ denotes the Euclidean length of the portion of π in f_i. Given two points a and b in P a path of minimum cost joining a and b is called *shortest path* between a and b

and is denoted by $a \overset{P}{\rightsquigarrow} b$. The cost of this path is referred to as *distance* between a and b and is denoted by $\mathrm{dist}_P(a, b)$.

Let ε be a real number in $(0, 1)$. Our algorithms construct an *approximation graph* $G_\varepsilon = (V_\varepsilon, E_\varepsilon)$, which is a supergraph of the corresponding graph in [6], whose nodes correspond to geometric objects, namely, Steiner points and vertex vicinities of "small" radius in P. Around each vertex v of P we define a "small" star-shaped polygon $\mathcal{E}(v)$ called *vertex vicinity*; it is contained within the union of the triangles incident to v and its intersections with each of the triangles is a "small" isosceles triangle with side length $\varepsilon r(v)$, where $r(v)$ is a fraction of the distance from v to the boundary of the union of the triangles incident to v. We set $r(v)$ to be (1/8)th of this distance (see, Definition 2.1 in [6]). The nodes of G_ε are of two types depending on the object in P they represent. Each vertex of P and its vertex vicinity is represented in G_ε as a node, called vertex vicinity node. Steiner point nodes represent Steiner points inserted in P. Steiner points are placed along the bisectors of the angles of the faces of P forming a geometric progressions with ratios depending on ε and on the geometry of P. The approximation properties of Steiner points are stated in the following lemma.

Lemma 2. *[6] Let x and y be points lying on two different edges of a face f of P and outside vertex vicinities. There is a Steiner point p in f such that $|xp| + |py| \le (1 + \varepsilon/2)|xy|$.*

Lemma 3. *[6]* **(a)** *The number of nodes in G_ε incident to a triangle f of P is bounded by $C(f)\frac{1}{\sqrt{\varepsilon}} \log \frac{2}{\varepsilon}$, where the constant $C(f)$ depends on the geometry[3] of the triangle f.* **(b)** *The total number of nodes of G_ε is less than $C(P)\frac{n}{\sqrt{\varepsilon}} \log \frac{2}{\varepsilon}$, where the constant $C(P) = \frac{1}{n}\sum_{f \in P} C(f)$.*

To define the edges of G_ε we introduce the notion of a *face neighborhood*. The face neighborhood of a vertex of P is the union of the triangles incident to that vertex. The face neighborhood, $\mathcal{N}(a)$, of a point a in a face f of P consists of the union of f and its *neighboring* faces. A node p of G_ε is connected to all nodes, whose representations are incident with its face neighborhood $\mathcal{N}(p)$. To define costs of the edges of G_ε we use the notion of *local paths*. A path in P is called *local* if it intersects at most two faces. The cost $c(p, q)$ of an edge (p, q) in G_ε is defined as the cost of the *local shortest path* restricted to lie in the intersection of their face neighborhoods $\mathcal{N}(p) \cap \mathcal{N}(q)$.

3.1 Approximation Properties of G_ε

The paths in the approximation graph G_ε are called *discrete paths*. The cost $c(\pi_G(p, q))$ of a discrete path $\pi_G(p, q)$ is the sum of the costs of its edges. For a pair of nodes p and q of G_ε, $p \overset{G}{\rightsquigarrow} q$ denotes a shortest path in G_ε between p and q. As is shown below, in general, the cost $c(p \overset{G}{\rightsquigarrow} q)$ of a shortest discrete

[3] Roughly it is about two times the sum of the reciprocals of the sinuses of the angles of f. See [6] for precise estimates.

path is an ε-approximation of the distance $\|\tilde{p} \overset{P}{\leadsto} \tilde{q}\|$, where $\tilde{p}$ and $\tilde{q}$ are points in P incident to the objects represented by p and q. A discrete path $\pi_G(p,q)$ between nodes p and q can be naturally embedded in P as follows. First, each node on that path is embedded into the object it represents, i.e. either a Steiner point or a vertex vicinity. Then each edge of $\pi_G(p,q)$ is embedded into the local shortest path between the objects representing its end-nodes. As a result we obtain a sequence of vertex vicinities joined by polygonal paths in P. Finally, we replace each vertex vicinity in $\pi_G(p,q)$ with a two segment path through the corresponding vertex of P. We refer to this embedding of a discrete path $\pi_G(p,q)$ into P as *natural embedding* and denote it by $\tilde{\pi}_G(p,q)$. By our definitions the cost in P of the natural embedding $\tilde{\pi}_G(p,q)$ of a discrete path minus the cost of its portions inside vertex vicinities equals to the cost of $\pi_G(p,q)$ in G_ε.

Theorem 3. *(follows from Theorem 4.5 in [6]) The SSSP problem in the approximation graph G_ε can be solved in $O(|V_\varepsilon| \log |V_\varepsilon|) = O(\frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$ time.*

Next, we discuss how the approximation graph G_ε can be used to approximate distances and shortest paths in P. Let a and b be arbitrary points in P. If a and b lie in neighboring triangles and the shortest path $a \overset{P}{\leadsto} b$ between them is a local path (i.e. stays inside the quadrilateral formed by the union of their triangles) than we can report the exact path in constant time. So, we concentrate on the approximation of shortest paths that cross more than two faces.

Naturally, we consider paths of the form $\{a \overset{P}{\leadsto} p \overset{G}{\leadsto} q \overset{P}{\leadsto} b\}$ and then approximate $\text{dist}_P(a,b)$ by the minimum of $\|a \overset{P}{\leadsto} p\| + c(p \overset{G}{\leadsto} q) + \|q \overset{P}{\leadsto} b\|$ taken over all choices of nodes p and q in G_ε. As it is shown below, we can obtain the desired approximation by taking the minimum not over all pairs of nodes in G_ε, but only over pairs p and q such that $p \in \mathcal{N}(a)$ and $q \in \mathcal{N}(b)$. Moreover we show that, it suffices to compute the local shortest paths between a and p and between b and q. We denote these local shortest paths by $a \overset{\mathcal{N}(a)}{\leadsto} p$ and $q \overset{\mathcal{N}(b)}{\leadsto} b$ and define *approximate discrete paths* between pairs of points in P as follows.

Definition 1. *A path between a pair of points a and b in P is called* approximate discrete path *if it is a shortest local path joining a and b or a path of the form*

$$\{a \overset{\mathcal{N}(a)}{\leadsto} p \overset{G}{\leadsto} q \overset{\mathcal{N}(b)}{\leadsto} b\}, \tag{1}$$

where $p \in \mathcal{N}(a)$, $q \in \mathcal{N}(b)$. The cost of an approximate discrete path is $\|a \overset{\mathcal{N}(a)}{\leadsto} p\| + c(p \overset{G}{\leadsto} q) + \|q \overset{\mathcal{N}(b)}{\leadsto} b\|$ or its cost in P if it is a local path. The cost of a shortest approximate discrete path between a and b is called approximate distance *between a and b and is denoted by* $\text{dist}_G(a,b)$.

Note that by this definition the approximate distance $\text{dist}_G(a,b)$ between points a and b lying in neighbor triangles is the minimum of the cost of the shortest local path between a and b and the cost of any path of the form (1). The next theorem establishes the relation between approximate distances and the distances in P.

Theorem 4. *For any pair of points a and b in P one of the following two holds: either* **(a)** $(1-2\varepsilon)\text{dist}_P(a,b) \leq \text{dist}_G(a,b) \leq (1+2\varepsilon)\text{dist}_P(a,b)$, *or* **(b)** $\text{dist}_P(a,b) - 2\varepsilon r(v) \leq \text{dist}_G(a,b) \leq \text{dist}_P(a,b)$, *where* $\varepsilon r(v)$ *is the radius of* $\mathcal{E}(v)$.

If case (a) of the theorem applies then the approximate distance $\text{dist}_G(a,b)$ is an approximation of the distance $\text{dist}_P(a,b)$ with relative error $\frac{|\text{dist}_G(a,b)-\text{dist}_P(a,b)|}{\text{dist}_P(a,b)}$ bounded by 2ε. Case (b) of the theorem can be viewed as an exception covering a special situation where points a and b are "close" to each other and "near" a vertex v of P, meaning that any shortest path in P between them has to intersect the vertex vicinity $\mathcal{E}(v)$ and must stay in the face neighborhood $\mathcal{N}(v)$. Furthermore, in case (b) $\text{dist}_G(a,b)$ is less than $\text{dist}_P(a,b)$ and it is an approximation with relative error not exceeding $\varepsilon r(v)/\text{dist}_G(a,b)$. Therefore, if $r(v) \leq 2\text{dist}_G(a,b)$ the quality of the approximation is 2ε, same as in case (a). If the ratio $r(v)/\text{dist}_G(a,b)$ is larger than $1/\varepsilon$ then the relative error could be as big as 1. For example, if points a and b are inside the vertex vicinity $\mathcal{E}(v)$ then $\text{dist}_G(a,b)$ is zero and the relative error is 1. Note, that the conditions for the occurrence of case (b) and the presence of eventually large (compared to ε) relative error are easily detected by the position of the points a and b, the structure of the approximate discrete path and the ratio $r(v)/\text{dist}_G(a,b)$. Thus if the approximation is not satisfactory the exact shortest path restricted to lie inside $\mathcal{N}(v)$ can be computed. This is summarized as follows.

Corollary 1. *The distance* $\text{dist}_G(a,b)$ *approximates* $\text{dist}_P(a,b)$ *with relative error* 2ε, *except possibly when the case (b) of Theorem 4 applies and* $r(v) > 2\text{dist}_G(a,b)$. *In the latter case* $\text{dist}_P(a,b)$ *can be computed directly.*

We conclude this section by a remark on the computation of approximate distances. The distance $\text{dist}_G(a,b)$ between a pair of points a and b can be computed as follows. We compute $\min(\|a \overset{\mathcal{N}(a)}{\rightsquigarrow} p\| + c(p \overset{G}{\rightsquigarrow} q) + \|q \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|)$, over all pairs of nodes $p \in \mathcal{N}(a)$ and $q \in \mathcal{N}(b)$. In the case where a and b do not lie in neighbor faces this minimum is the approximate distance $\text{dist}_G(a,b)$. In the case where the points a and b lie in neighbor faces we also need to consider the shortest local path between them.

4 Fixed Source Shortest Path Queries

In this section we describe an algorithm, that takes as input a point a in P, called *source*, and an approximation parameter $\varepsilon \in (0,1)$ and constructs a data structure, called *Single Source Queries* (SSQ), such that for any query point $b \in P$, called *target*, the approximate distance $\text{dist}_G(a,b)$ (and/or an approximate shortest path) from a to b is computed efficiently. The algorithm uses the approximation graph G_ε. For simplicity, we assume that the point a corresponds to a node in G_ε; otherwise, we can easily augment G_ε with extra edges corresponding to local shortest paths from a to nodes in its face neighborhood $\mathcal{N}(a)$. Approximate discrete path between the node a and a point b is either a local shortest path (that can be computed in constant time) or a path of the form $\{a \overset{G}{\rightsquigarrow} p \overset{\mathcal{N}(b)}{\rightsquigarrow} b\}$, where p is a node of G_ε incident to the face neighborhood of b. Hence, the computation of $\text{dist}_G(a,b)$ requires finding $\min_{p\in\mathcal{N}(b)}\{\text{dist}_G(a,p) + \|p \overset{\mathcal{N}(b)}{\rightsquigarrow} b\|\}$. We know the distances $\text{dist}_G(a,p)$ from a to all nodes $p \in G_\varepsilon$ (as part of preprocessing by computing SSSP tree rooted at a in G_ε) and thus our task is reduced to finding a node $p(b)$ that minimizes the above expression for a query point b. To accomplish this, for each face f of P we construct a data structure, called *Local Voronoi Diagram* in f with respect to a, and denote it by $\text{LVD}(a,f)$. More precisely, let f be a face of P and let $\mathcal{N}(f)$ be its face neighborhood. Let $p_1,\ldots,p_k$ be the Steiner points and the vertices of P incident to $\mathcal{N}(f)$ and let $\delta_i = \text{dist}_G(a,p_i)$ for $i = 1,\ldots,k$.

Lemma 4. *A data structure* $\text{LVD}(a,f)$ *exists so that for a point* $b \in f$, $\min_{1\le i\le k}(\delta_i + \|b \overset{\mathcal{N}(b)}{\rightsquigarrow} p_i\|)$ *and the point for which it is achieved can be computed in* $O(\log k)$ *time. The size of* $\text{LVD}(a,f)$ *is* $O(k)$ *and it can be constructed in* $O(k\log k)$ *time.*

We define $\text{SSQ}(P,\varepsilon;a)$ data structure to consist of SSSP tree rooted at a plus the collection of $\text{LVD}(a,f)$ for all faces $f \in P$. The queries consist of a point b on P and the face $f(b)$ containing b and they are answered as follows. First, we use $\text{LVD}(a,f(b))$ and find the point $p(b)$ for which the minimum in Lemma 4 is achieved. Then, if a and b lie in neighbor faces, we compute the shortest local path between a and b and output the approximate distance $\text{dist}_G(a,b)$, which is the smaller of the two values. If an approximation path is required we output the natural embedding of the approximate discrete path whose cost is $\text{dist}_G(a,b)$. The quality of this approximation follows from Theorem 4 and Corollary 1. Hence we have the following

Theorem 5. *Given a triangulated, weighted surface* P *with* n *faces, a source point* $a \in P$ *and the set of nodes* V_ε *of* G_ε. *(For every query point* b *in* P *we assume that the face containing* b *is known.) A data structure* $\text{SSQ}(P,\varepsilon;a)$ *of size* $O(|V_\varepsilon|) = O(\frac{n}{\sqrt{\varepsilon}}\log\frac{1}{\varepsilon})$ *exists, so that the approximate distance between* a *and a query point* b *in* P *can be found in* $O(\log\frac{1}{\varepsilon})$ *time. The structure SSQ can be constructed in* $O(|V_\varepsilon|\log|V_\varepsilon|) = O(\frac{n}{\sqrt{\varepsilon}}\log\frac{n}{\varepsilon}\log\frac{1}{\varepsilon})$ *time.*

5 Arbitrary Shortest Path Queries

Next, we describe and analyze an algorithm for constructing a data structure, called *All Pairs Queries* (APQ), such that approximate distance (and/or approximate shortest path) queries between pairs of arbitrary points in P can be answered efficiently. In addition to the weighted polyhedral surface P and the approximation parameter ε, the algorithm takes as input a query time parameter $\mathfrak{q}$ and outputs a data structure $\text{APQ}(P,\varepsilon;\mathfrak{q})$, which can answer approximate distance queries in $O(\mathfrak{q})$ time. We build upon the results of previous sections.

First we construct the dual graph P^* of P. The set of nodes of P^* corresponds to the set of faces of P and two nodes in P^* are joined by an edge if their corresponding faces are neighbors. To each node u of P^* we assign weight $w(u)$ equal to the number of nodes of G_ε, that are incident to the face $f(u)$ corresponding to u in P. Furthermore, we assign cost $c(u)$ equal to the number of nodes of G_ε, that are incident to the face neighborhood $\mathcal{N}(f(u))$. The total weight $w(P^*)$ of P^* and the value $\sigma(P^*)$, defined in Section 2, are estimated using Lemma 3 by $w(P^*) \leq C(P)\frac{n}{\sqrt{\varepsilon}} \log\frac{2}{\varepsilon}$ and $\sigma(P^*) \leq \Gamma(P)\frac{n}{\varepsilon}\log^2\frac{2}{\varepsilon}$, where $C(P) = \frac{1}{n}\sum_{f\in P} C(f)$ and $\Gamma(P) \leq \frac{4}{n}\sum_{f\in P} C^2(f)$. We observe that $w(P^*)$ and $\sigma(P^*)$ are related by $\sigma(P^*) \leq 4w^2(P^*) \leq n\sigma(P^*)$.

Next, we choose a value of $t = \frac{\mathfrak{q}^2}{4(g+1)\sigma(P^*)\log^2(1/\varepsilon)}$ (depending on the input query time $\mathfrak{q}$) and use Theorem 2 to construct a t-separator S, that induces regular partitioning of P^*. The separator S of P^* corresponds to a set of faces in P, which we refer to as *face separator* (or simply separator) and denote again by S. The face separator S partitions the surface P into *regions*, that are unions of faces corresponding to the connected components of $P^* \setminus S$. The boundary ∂R of a region R is the set of triangles in S, that neighbor faces in R.

Next, for each p, that is a Steiner point or vertex of P incident to a face, which is a neighbor of a face in S, compute and store $\mathrm{SSQ}(P,\varepsilon;p)$ data structure. For each region R and for each Steiner point or vertex of P incident to a face in R compute and store $\mathrm{SSQ}(R,\varepsilon;p)$ data structure restricted to the faces in R.

The collection of SSQ data structures and the region partitioning induced by S constitutes the $\mathrm{APQ}(P,\varepsilon;\mathfrak{q})$ data structure. We denote the genus of the surface P by g. The query time parameter $\mathfrak{q}$ will not exceed an upper bound $\bar{\mathfrak{q}} = \frac{(g+1)^{2/3}n^{1/3}}{\sqrt{\varepsilon}}\log^2\frac{1}{\varepsilon}$. The next lemma presents our estimate on the time for the construction and the size of $\mathrm{APQ}(P,\varepsilon;\mathfrak{q})$ data structure.

Lemma 5. *For any* $\mathfrak{q} \leq \bar{\mathfrak{q}} = \frac{(g+1)^{2/3}n^{1/3}}{\sqrt{\varepsilon}}\log^2\frac{1}{\varepsilon}$ *the construction of the data structure* $\mathrm{APQ}(P,\varepsilon;\mathfrak{q})$ *takes* $O(\frac{(g+1)n^2}{\varepsilon^{3/2}\mathfrak{q}}\log\frac{n}{\varepsilon}\log^4\frac{1}{\varepsilon})$ *time. The size of* APQ $(P,\varepsilon;\mathfrak{q})$ *is* $O(\frac{(g+1)n^2}{\varepsilon^{3/2}\mathfrak{q}}\log^4\frac{1}{\varepsilon})$.

The APQ data structure built by the preprocessing algorithm can be used to answer approximate distance queries as outlined in Algorithm APQ_Query. Note that set A plays a critical role in the query algorithm. A set of nodes in G_ε is called a *separating set* for points a and b in P if any approximate discrete path of the form (1) between a and b contains a node from that set. Our query algorithm specifies a separating set A for a and b, such that for any $p' \in A$ the data structure $\mathrm{SSQ}(P,\varepsilon;p')$ is present in $\mathrm{APQ}(P,\varepsilon;\mathfrak{q})$ and then computes $\min_{p'\in A}(\mathrm{dist}_G(a,p') + \mathrm{dist}_G(p',b))$. Clearly, this minimum is the cost of the shortest approximate discrete path of the form (1). The time for this computation is $O(|A|\log\frac{1}{\varepsilon})$.

If an approximate shortest path between a and b is required we output the natural embedding of the approximate discrete path for which $\mathrm{dist}_G(a,b)$ is achieved. This can be done by using the SSSP trees stored in the corresponding SSQ data structure in time proportional to the size of this path. The next lemma establishes the correctness and running time of the query algorithm.

```
ALGORITHM: APQ_Query
Input: Data structure APQ($P, \varepsilon; \mathfrak{q}$); query points $a$ and $b$ lying in
    faces $f(a)$ and $f(b)$, respectively.
Output: Approximate distance $\mathrm{dist}_G(a, b)$.
Set $M_0 = M_1 = M_2 = \infty$.
Step 1. If $f(a)$ and $f(b)$ are neighbor faces, then compute the local shortest path
    $a \overset{f(a) \cup f(b)}{\rightsquigarrow} b$ and assign its cost to $M_0$.
Step 2. If either of the faces $f(a)$ or $f(b)$ is in the separator $S$,
    then define $A$ to be the set of nodes of $G_\varepsilon$ incident to the face
    neighborhood $\mathcal{N}(a)$ or $\mathcal{N}(b)$, respectively.
Step 3. If neither of the faces $f(a)$ and $f(b)$ is in $S$,
    then define $A$ to be the set of nodes of $G_\varepsilon$ incident to the faces
    in the boundary $\partial R(a)$ of the region $R(a)$ containing $f(a)$.
Step 4. Use data structures $SSQ(P, \varepsilon; p')$ and compute
    $M_1 = \min_{p' \in A}(\mathrm{dist}_G(a, p') + \mathrm{dist}_G(p', b))$.
Step 5. If $f(b) \in R(a)$ then define $A_1$ to be the set of nodes of $G_\varepsilon$
    incident to the face neighborhood $\mathcal{N}(b)$. Use data structures $SSQ(R, \varepsilon; p')$
    and compute $M_2 = \min_{p' \in A_1}(\mathrm{dist}_G(a, p') + \mathrm{dist}_G(p', b))$.
Set $\mathrm{dist}_G(a, b) = \min(M_0, M_1, M_2)$ and output it.
```

Lemma 6. *The algorithm APQ_Query correctly computes the approximate distance* $\mathrm{dist}_G(a, b)$. *The running time of the algorithm is* $O(\max(\mathfrak{q}, \frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon})$.

Proof: The correctness of the query algorithm follows easily. The running time of the algorithm is dominated by the times for the execution of Steps 4 and 5. As discussed above these times are bounded by $O(|A| \log \frac{1}{\varepsilon})$ and $O(|A_1| \log \frac{1}{\varepsilon})$. By Lemma 3, $|A_1| = O(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$ and by Theorem 2 and the choice of t in the APQ_Preprocessing algorithm we obtain $|A| \leq 2\sqrt{(g+1)t\sigma(P^*)} \leq \frac{\mathfrak{q}}{\log(1/\varepsilon)}$. $\square$

Theorem 6. *Let P be a weighted polyhedral surface consisting of n triangular faces of genus g. Let $\varepsilon \in (0, 1)$ and $\mathfrak{q} \in (\frac{1}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon}, \bar{\mathfrak{q}})$, where $\bar{\mathfrak{q}} = \frac{(g+1)^{2/3} n^{1/3}}{\sqrt{\varepsilon}} \log^2 \frac{1}{\varepsilon}$. There exists a data structure* APQ($P, \varepsilon; \mathfrak{q}$), *such that approximate distance queries in P can be answered in $O(\mathfrak{q})$ time. The size of* APQ($P, \varepsilon; \mathfrak{q}$) *is* $O(\frac{(g+1)n^2}{\varepsilon^{3/2}\mathfrak{q}} \log^4 \frac{1}{\varepsilon})$ *and it is constructed in* $O(\frac{(g+1)n^2}{\varepsilon^{3/2}\mathfrak{q}} \log \frac{n}{\varepsilon} \log^4 \frac{1}{\varepsilon})$ *time.*

6 Extensions and Conclusions

In this paper we present novel solutions to fundamental shortest path query problems. The algorithms improve and generalize previous solutions in terms of 1) setting: a) Euclidean to weighted and b) arbitrary genus g, 2) preprocessing time, and/or 3) size of query data structure. We also develop a new graph partitioning algorithm for graphs of genus g with weights and costs on vertices which extends and/or generalizes previously known separator algorithms. Our techniques also enable us to obtain improved results with space-query time tradeoffs for the planar case, i.e. when the genus is 0 (see the full version of this paper). By constructing a hierarchical APQ data structure, in which second level APQ data structures are built and stored for each region of the partitioning we can widen the range of query time parameter $\mathfrak{q}$, in Theorem 6, while keeping the efficiency of the algorithm. Our analysis shows that the result of Theorem 6 extends to the case $\mathfrak{q} \in (\bar{\mathfrak{q}}, \bar{\mathfrak{q}}_1)$, where $\bar{\mathfrak{q}}_1 = \frac{(g+1)^{5/9} n^{2/3}}{\varepsilon^{2/3}} \log^{7/3} \frac{1}{\varepsilon}$. Our technique can be used to build an APQ data structure with query time parameter $\mathfrak{q} = \log \frac{1}{\varepsilon}$. This is achieved by building a suitable ε-mesh consisting of $O(1/\varepsilon^2)$ additional points in each triangle f of P. Note that our algorithms inherit the geometric constants analyzed in [6]. It is an interesting open problem to determine whether eliminating these constants is inherently impossible.

References

1. P. K. Agarwal, B. Aronov, J. O'Rourke, and C. A. Schevon. Star unfolding of a polytope with applications. *SIAM J. Comput.*, 26(6):1689–1713, 1997.
2. P. K. Agarwal, S. Har-Peled, M. Sharir, and K.R. Varadarajan. Approximate shortest paths on a convex polytope in three dimensions. *J.ACM*, 44:567–584, March 1997.
3. L. Aleksandrov and H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Disc. Math.*, 9(1):129–150, 1996.
4. L. Aleksandrov, H. Djidjev, H. Guo, and A. Maheshwari. Partitioning planar graphs with costs and weights. In *ALENEX '02: Revised Papers from the 4th International Workshop on Algorithm Engineering and Experiments*, pages 98–110, London, UK, 2002. Springer-Verlag.
5. L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ε-approximation algorithm for weighted shortest path queries on polyhedral surfaces. In *Proc. 14th Euro CG Barcelona*, pages 19–21, 1998.
6. L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *J. ACM*, 52(1):25–53, 2005.
7. B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. *SIAM J. Comput.*, 35:627–646, 2006.
8. J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proc. 6th ACM Symposium on Computational Geometry*, pages 360–369, 1990. also IJCGA 6: 127-144, 1996.
9. Y.-J. Chiang and J. S. B. Mitchell. Two-point euclidean shortest path queries in the plane. In *Proc. 10th ACM-SODA*, pages 215–224, Philadelphia, USA, 1999.
10. H. N. Djidjev. Linear algorithms for graph separation problems. In *SWAT'88, LNCS*, volume 318, pages 643–645. Springer-Verlag, Berlin, Heidelberg, 1988.
11. G. N. Frederickson. Fast algorithms for shortest paths in planar graphs. *SIAM J. Comput.*, 16:1004–1022, 1987.
12. J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5:391–407, 1984.
13. S. Har-Peled. Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. *DCG*, 21:216–231, 1999.
14. S. Har-Peled. Constructing approximate shortest path maps in three dimensions. *SIAM J. Comput.*, 28(4):1182–1197, 1999.

A Unified Construction of the Glushkov, Follow, and Antimirov Automata

Cyril Allauzen and Mehryar Mohri

Courant Institute of Mathematical Sciences
251 Mercer Street, New York, NY 10012, USA
`{allauzen, mohri}@cs.nyu.edu`
`http://www.cs.nyu.edu/~{allauzen, mohri}`

Abstract. A number of different techniques have been introduced in the last few decades to create ϵ-free automata representing regular expressions such as the *Glushkov automata*, *follow automata*, or *Antimirov automata*. This paper presents a simple and unified view of all these construction methods both for unweighted and weighted regular expressions. It describes simpler algorithms with time complexities at least as favorable as that of the best previously known techniques, and provides a concise proof of their correctness. Our algorithms are all based on two standard automata operations: epsilon-removal and minimization. This contrasts with the multitude of complicated and special-purpose techniques previously described in the literature, and makes it straightforward to generalize these algorithms to the weighted case. In particular, we extend the definition and construction of follow automata to the case of weighted regular expressions over a closed semiring and present the first algorithm to compute weighted Antimirov automata.

1 Introduction

The construction of finite automata representing regular expressions has been widely studied due to their multiple applications to pattern-matching and many other areas of text processing [1,21]. The most classical construction, Thompson's construction [13,24], creates a finite automaton with a number of states and transitions linear in the length m of the regular expression. The time complexity of the algorithm is also linear, $O(m)$. But Thompson's automaton contains transitions labeled with the empty string ϵ which create a delay in pattern matching. Many alternative techniques have been introduced in the last few decades to create ϵ-free automata representing regular expressions, in particular, Glushkov automata [11], follow automata [12], and Antimirov automata [2].

The Glushkov automaton, or position automaton, was independently introduced by [11] and [16]. It has exactly $n+1$ states but may have up to n^2 transitions, where n is the number of occurrences of alphabet symbols appearing in the expression. Thus, its size is quadratic in that of the Thompson automaton for reasonable regular expressions for which $m = O(n)$. However, when using bit-parallelism for regular expression search, thanks to its smaller number of states, the Glushkov automaton can be represented with half the number of machine words required by the Thompson automaton [20,21].

Table 1. Simple algorithms for the construction of Glushkov, follow, and Antimirov automata and their time complexity. T is the Thompson automaton. For an automaton A, $\overline{A}$ denotes the automaton derived from A by marking alphabet symbols with their position in the expression. When the symbols are marked, the same notation denotes the operation that removes the marking. $\widehat{T}$ is obtained by marking some ϵ-transitions in T, making it deterministic (the ϵ-transitions marked are removed by the $\widehat{\text{rmeps}}$ operation).

Automaton	Algorithm	Complexity
Glushkov	$\text{rmeps}(T)$	$O(mn)$
Follow	$\overline{\min(\text{rmeps}(\overline{T}))}$	$O(mn)$
Antimirov	$\widehat{\text{rmeps}}(\min(\text{rmeps}(\widehat{T})))$	$O(m \log m + mn)$

Several techniques have been suggested for constructing the Glushkov automaton. In [3], the construction is based on the recursive definition of the follow function and its complexity is in $O(n^3)$. The algorithm described by [4] is based on an optimization of the recursive definition of the follow function and its complexity is in $O(m + n^2)$. It requires the expression to be first rewritten in star-normal form, which can be done non-trivially in $O(m)$. Several other quadratic algorithms have been given: that of [9] which is based on an optimization of the follow recursion, and that of [22], based on the ZPC structure, which consists of two mutually linked copies of the syntactic tree of the expression.

The Antimirov or partial derivatives automaton was introduced by [2]. It is in general smaller than the Glushkov automaton with up to $n+1$ states and up to n^2 transitions. It was in fact proven by [8] (see [12] for a simpler proof) to be the quotient of the Glushkov automaton for some equivalence relation. The complexity of the original construction algorithm of [2] is $O(m^5)$. [8] presented an algorithm whose complexity is $O(m^2)$.

Finally, the follow automaton was introduced by [12]. It is the quotient of the Glushkov automaton by the *follow equivalence*: two states are equivalent if they have the same follow and the same finality. The author gave an $O(m + n^2)$ algorithm where some ϵ-transitions are removed from the automaton at each step of the Thompson construction as well as at the end. An $O(m + n^2)$ algorithm using the ZPC structure was given in [7], which requires the regular expression to be rewritten in star-normal form.

Some of these results have been extended to weighted regular expressions over arbitrary semirings. The generalization of the Thompson construction trivially follows from [23]. The Glushkov automaton can be naturally extended to the weighted case [5], and an $O(m^2)$ construction algorithm based on the generalization of the ZPC construct was given by [6]. The Antimirov automaton was generalized to the weighted case by [15], but no explicit construction algorithm or complexity analysis was given by the authors.

This paper presents a simple and unified view of all these ϵ-free automata (Glushkov, follow, and Antimirov) both in the case of unweighted and weighted regular expressions. It describes simpler algorithms with time complexities at least as favorable as that of the best previously known techniques, and provides concise proofs. Our algorithms are all based on two standard automata operations: epsilon-removal and minimization, as summarized in Table 1.

This contrasts with the multitude of complicated and special-purpose techniques and proofs described by others to construct these automata: no need for fine-tuning some recursions, no requirement that the regular expression be in star-normal form, and no need to maintain multiple copies of the syntactic tree. Our analysis provides a better understanding of ϵ-free automata representing regular expressions: they are all the results of the application of some combinations of epsilon-removal and minimization to the classical Thompson automata. This makes it straightforward to generalize these algorithms to the weighted case by using the generalization of ϵ-removal and minimization [17,18]. It also results in much simpler algorithms than existing ones.

In particular, this leads to a straightforward algorithm for the construction of the Glushkov automaton of a weighted regular expression, and, in the case of closed semirings, helps us generalize follow automata to the weighted case. We also give the first explicit construction algorithm of the Antimirov automaton of a weighted expression. When the semiring is k-closed, or, in the Glushkov case, only null-k-closed for the regular expression considered, the complexity of our algorithms is the same as in the unweighted case.

The paper is organized as follows. Section 2 introduces the definitions and a brief description of the elementary algorithms used in the following sections. Section 3 presents and analyzes our algorithm for the construction of the Glushkov automaton, Section 4 the same for the follow automaton, and Section 5 for the Antimirov automaton.

2 Preliminaries

Semirings. A system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a *semiring* when $(\mathbb{K}, \oplus, \overline{0})$ is a commutative monoid with identity element $\overline{0}$; $(\mathbb{K}, \otimes, \overline{1})$ is a monoid with identity element $\overline{1}$; $\otimes$ distributes over $\oplus$; and $\overline{0}$ is an annihilator for $\otimes$: for all $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$. Thus, a semiring is a ring that may lack negation. Some familiar semirings include the boolean semiring $(\mathbb{B}, \vee, \wedge, 0, 1)$, the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$, and the real semiring $(\mathbb{R}_+, +, \times, 0, 1)$.

A semiring $\mathbb{K}$ is said to be *closed* if for all $a \in \mathbb{K}$, the infinite sum $\bigoplus_{n=0}^{\infty} a^n$ is well-defined and in $\mathbb{K}$, and if associativity, commutativity, and distributivity apply to countable sums [19]. $\mathbb{K}$ is said to be k-closed if for all $a \in \mathbb{K}$, $\bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n$. More generally, we will say that $\mathbb{K}$ is *closed* (*k-closed*) *for an automaton* A, if the closedness (resp. k-closedness) axioms hold for all cycle weights of A. In some semirings, e.g., the probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$, the equality $\bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n$ may hold for the cycle weights of A only approximately, modulo $\epsilon > 0$. A is then said to be *ϵ-k-closed* for that semiring.

Weighted automata. A *weighted automaton* A over a semiring $\mathbb{K}$ is a 7-tuple $(\Sigma, Q, E, I, \lambda, F, \rho)$ where: Σ is a finite alphabet; Q is a finite set of states; $I \subseteq Q$ the set of initial states; $F \subseteq Q$ the set of final states; $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$ a finite set of transitions; $\lambda : I \to \mathbb{K}$ the initial weight function; and $\rho : F \to \mathbb{K}$ the final weight function mapping F to $\mathbb{K}$.

We denote by $p[\pi]$ the origin and $n[\pi]$ the destination state of a path $\pi \in E^*$ in an automaton A. $i[\pi]$ denotes the label of π, and $w[\pi]$ its weight obtained by $\otimes$-multiplying the weights of its constituent transitions. We also denote by $P(p, q)$

the set of paths from p to q and by $P(I, x, F)$ the set of paths from the initial states I to the final states F labeled with $x \in \Sigma^*$. The weight associated by A to an input string $x \in \Sigma^*$ is obtained by summing the weights of these paths multiplied by their initial and final weights: $[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$. $[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I, x, F) = \emptyset$.

Shortest distance. Let A be a weighted automaton over $\mathbb{K}$. The shortest distance from p to q is defined as $d[p, q] = \bigoplus_{\pi \in P(p,q)} w[\pi]$. It can be computed using the generic single-source shortest-distance algorithm of [19] if $\mathbb{K}$ is k-closed for A, or using a generalization of Floyd-Warshall [14,19] if $\mathbb{K}$ is closed for A.

Epsilon-removal. The general ϵ-removal algorithm of [18] consists of first computing the ϵ-closure of each state p in A,

$$\text{closure}(p) = \{(q, w) \colon w = d_\epsilon[p, q] = \bigoplus_{\pi \in P(p,q), i[\pi]=\epsilon} w[\pi] \neq \overline{0}\}, \tag{1}$$

and then, for each state p, of deleting all the outgoing ϵ-transitions of p, and adding out of p all the non-ϵ transitions leaving each state $q \in \text{closure}(p)$ with their weight pre-$\otimes$-multiplied by $d_\epsilon[p, q]$. If $\mathbb{K}$ is k-closed for the ϵ-cycles of A,[1] then the generic single-source shortest-distance algorithm [19] can be used to compute the ϵ-closures.

Weight-pushing and weighted minimization. Weight-pushing [17] is a normalization algorithm that redistributes the weights along the paths of A such that $\bigoplus_{e \in E[q]} w[e] + \rho(q) = \overline{1}$ for every state $q \in Q$. We denote by push(A) the resulting automaton. The algorithm requires that $\mathbb{K}$ be zero-sum free, weakly left-divisible and closed or k-closed for A since it depends on the computation of $d[q, F]$ for all $q \in Q$. It was proved in [17] that, if A is deterministic, *i.e.*, if no two transitions leaving any state share the same label and if it has a unique initial state, then the weight-pushing followed by unweighted minimization with each pair (label, weight) viewed as an alphabet symbol, leads to a minimal deterministic weighted automaton equivalent to A, denoted by min(A).

Regular expressions. A *weighted regular expression* over the semiring $\mathbb{K}$ is defined recursively by: $\emptyset$, ϵ and $a \in \Sigma$ are regular expressions, and if α and β are regular expressions then $k\alpha$, αk for $k \in \mathbb{K}$, $\alpha + \beta$, $\alpha \cdot \beta$ and α^* are also regular expressions. We denote by null(α) the weight associated by α to the empty string ϵ. A weighted regular expression α is well-defined iff for every subterm of the form β^*, null$(\beta)^*$ is well-defined and in $\mathbb{K}$. We will say that $\mathbb{K}$ is *null-k-closed for* α if there exist $k \geq 0$ such that for every subterm β^* of α, $\text{null}(\beta)^* = \bigoplus_{i=0}^{k} \text{null}(\beta)^i$. We denote by $|\alpha|$ the *length of* α, and by $|\alpha|_\Sigma$ the *width of* α, *i.e.*, the number of occurrences of alphabet symbols in α. Let $\text{pos}(\alpha) = \{1, 2, \ldots, |\alpha|_\Sigma\}$ be the set of (alphabet symbol) positions in α. An unweighted regular expression can be seen as a weighted expression over the boolean semiring $(\mathbb{B}, \vee, \wedge, 0, 1)$.

Thompson automaton. We denote by $A_T(\alpha)$ the *Thompson automaton* of α and by $I_{A_T(\alpha)}$ and $F_{A_T(\alpha)}$ its unique initial and final states. For $i \in \text{pos}(\alpha)$, we denote by p_i and q_i the states of $A_T(\alpha)$ such that the transition from p_i to q_i is labeled with the alphabet symbol at the i-th position in α. The states p_i (states q_i) are the only states having a non-ϵ outgoing (resp. incoming) transition.

[1] For A to be well-defined, $\mathbb{K}$ needs to be closed for the ϵ-cycles of A.

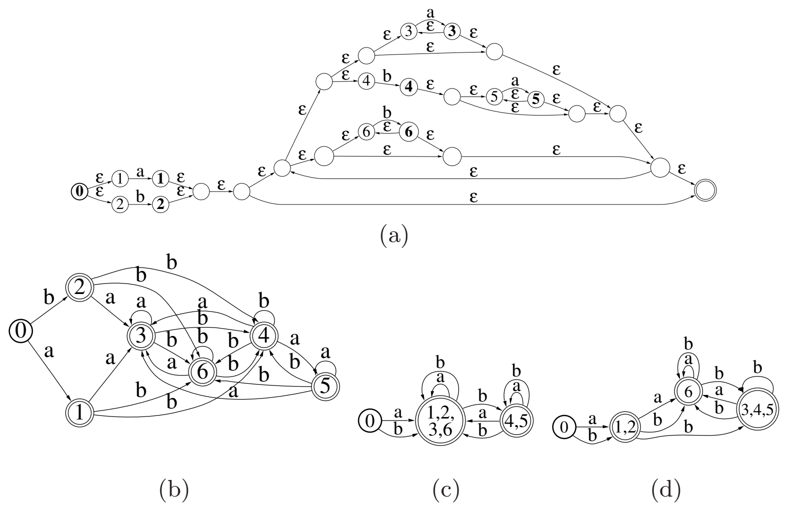


Fig. 1. (a) The Thompson automaton, (b) Glushkov automaton, (c) Follow automaton, and (d) Antimirov automaton of the regular expression $\alpha = (a + b)(a^* + ba^* + b^*)^*$, the running example used in [12]. In (d), state $\{0\}$ corresponds to the derived term α, $\{1, 2\}$ to $\tau = (a^* + ba^* + b^*)^*$, $\{6\}$ to $a^*\tau$, and $\{3, 4, 5\}$ to $b^*\tau$.

Figure 1(a) shows the Thompson automaton in the special case of the regular expression $\alpha = (a + b)(a^* + ba^* + b^*)^*$.

3 Glushkov Automaton

Let α be a weighted regular expression over the alphabet Σ and the semiring $\mathbb{K}$. The Glushkov automaton of α is an ϵ-free non-deterministic weighted automaton representing α that has an initial state plus one state for each position in α, *i.e.* each occurrence of an alphabet symbol in α. Figure 1(b) shows an example.

The formal definition of the Glushkov automaton is based on the functions null, first, last, and follow. Table 2 gives the recursive definition of these functions. null$(\overline{\alpha}) \in \mathbb{K}$ is the weight associated by $\overline{\alpha}$ to the empty string ϵ and is thus the final weight of the initial state of the automaton. last$(\overline{\alpha}) \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ is the set of positions with the corresponding final weights where a non-empty string accepted by α can end. first$(\overline{\alpha}) \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ is the set of positions with the corresponding weights that can be reached by reading one alphabet symbol from the initial state. Similarly, follow$(\overline{\alpha}, i) \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ is the set of positions with the corresponding weights that can be reached by reading one alphabet symbol from the position i.

In these definitions, the union of two weighted subsets $X, Y \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ is defined by $X \cup Y = \{(i, \langle X, i\rangle \oplus \langle Y, i\rangle) : \langle X, i\rangle \oplus \langle Y, i\rangle \neq \overline{0}\}$. For any weighted subset $X \subseteq \text{pos}(\alpha) \times \mathbb{K}$, weight $k \in \mathbb{K}$, and position i, $k \otimes X$ denotes the weighted subset and $\langle X, i\rangle$ the weight defined by:

$$k \otimes X = \begin{cases} \{(i, k \otimes w) | (i, w) \in X\} & \text{if } k \neq \overline{0}, \\ \emptyset & \text{otherwise,} \end{cases} \quad \text{and } \langle X, i\rangle = \begin{cases} w \text{ if } \exists\, w\colon (i, w) \in X, \\ \overline{0} \text{ otherwise.} \end{cases}$$

Table 2. Definition of the functions null, first, last, and follow. For convenience, we also define $\text{follow}(\overline{\alpha}, 0) = \text{first}(\overline{\alpha})$ and $\text{last}_0(\overline{\alpha})$ as $\text{last}(\overline{\alpha}) \cup \{(0, \text{null}(\overline{\alpha}))\}$ if $\text{null}(\overline{\alpha}) \neq \overline{0}$, $\text{last}(\overline{\alpha})$ otherwise.

$\overline{\alpha}$	$\text{null}(\overline{\alpha})$	$\text{first}(\overline{\alpha})$	$\text{last}(\overline{\alpha})$	$\text{follow}(\overline{\alpha}, i)$
$\emptyset$	$\overline{0}$	$\emptyset$	$\emptyset$	$\emptyset$
ϵ	$\overline{1}$	$\emptyset$	$\emptyset$	$\emptyset$
a_j	$\overline{0}$	$\{(j, \overline{1})\}$	$\{(j, \overline{1})\}$	$\emptyset$
$k\beta$	$k \otimes \text{null}(\beta)$	$k \otimes \text{first}(\beta)$	$\text{last}(\beta)$	$\text{follow}(\beta, i)$
βk	$\text{null}(\beta) \otimes k$	$\text{first}(\beta)$	$\text{last}(\beta) \otimes k$	$\text{follow}(\beta, i)$
$\beta + \gamma$	$\text{null}(\beta) \oplus \text{null}(\gamma)$	$\text{first}(\beta) \cup \text{first}(\gamma)$	$\text{last}(\beta) \cup \text{last}(\gamma)$	$\begin{cases} \text{follow}(\beta, i) \text{ if } i \in \text{pos}(\beta) \\ \text{follow}(\gamma, i) \text{ if } i \in \text{pos}(\gamma) \end{cases}$
$\beta \cdot \gamma$	$\text{null}(\beta) \otimes \text{null}(\gamma)$	$\text{null}(\beta) \otimes \text{first}(\gamma) \cup \text{first}(\beta)$	$\text{last}(\beta) \otimes \text{null}(\gamma) \cup \text{last}(\gamma)$	$\begin{cases} \text{follow}(\beta, i) \cup \langle \text{last}(\beta), i \rangle \otimes \text{first}(\gamma) \text{ if } i \in \text{pos}(\beta) \\ \text{follow}(\gamma, i) \text{ if } i \in \text{pos}(\gamma) \end{cases}$
β^*	$\text{null}(\beta)^*$	$\text{null}(\beta)^* \otimes \text{first}(\beta)$	$\text{last}(\beta) \otimes \text{null}(\beta)^*$	$\text{follow}(\beta, i) \cup \langle \text{last}(\beta^*), i \rangle \otimes \text{first}(\gamma)$

$X \otimes k$ is defined similarly. Let $\overline{\alpha}$ denote the weighted regular expression obtained by marking each symbol of α with its position. The *Glushkov* or *position automaton* $A_G(\alpha)$ *of* α is defined by $A_G(\alpha) = (\Sigma, \text{pos}_0(\alpha), E, 0, \overline{1}, F, \rho)$ where its states set is $\text{pos}_0(\alpha) = \{0\} \cup \text{pos}(\alpha)$ and its transition set

$$E = \{(i, a, w, j) \colon (j, w) \in \text{follow}(\overline{\alpha}, i) \text{ and } \text{pos}(\alpha, j) = a\}. \tag{2}$$

A state $i \in \text{pos}_0(\alpha)$ is final iff there exist $w \in \mathbb{K}$ such that $(i, w) \in \text{last}_0(\overline{\alpha})$ and when it is final $\rho(i) = w$. The following lemma shows that there exists a simple relationship between the first, last, and follow functions and the ϵ-closures of the states in the Thompson automaton that admit a non-ϵ incoming transition (states q_i).

Lemma 1. *Let α be a weighted regular expression and let $A = A_T(\alpha)$. Then*

(i) $(i, w) \in \text{first}(\overline{\alpha})$ *iff* $(p_i, w) \in \text{closure}(I_A)$*;*
(ii) $(i, w) \in \text{follow}(\overline{\alpha}, j)$ *iff* $(p_i, w) \in \text{closure}(q_j)$*; and*
(iii) $(i, w) \in \text{last}(\overline{\alpha})$ *iff* $(F_A, w) \in \text{closure}(q_i)$*.*

Proof. Note that if $\alpha = a$, $\alpha = \epsilon$ or $\alpha = \emptyset$, then the properties trivially hold. The proof is by induction on the length of the regular expression and is given in the case $\alpha = \beta \cdot \gamma$. Other cases can be treated similarly.

Assume that the properties hold for all expressions shorter than α. Let $A = A_T(\alpha)$, $B = A_T(\beta)$ and $C = A_T(\gamma)$. If $\alpha = \beta \cdot \gamma$, then $\text{closure}_A(I_A) = \text{closure}_B(I_B) \cup [\![B]\!][\epsilon] \otimes \text{closure}_C(I_A)$, thus, since $[\![B]\!][\epsilon] = \text{null}(\beta)$, (i) holds by induction. If $j \in \text{pos}(\gamma)$, then $\text{closure}_A(q_j) = \text{closure}_C(q_j)$. Otherwise, if $j \in \text{pos}(\beta)$, then

$$\text{closure}_A(q_j) = \text{closure}_B(q_j) \cup \langle \text{closure}_B(q_j), F_B \rangle \otimes \text{closure}_C(I_C). \tag{3}$$

Thus, by induction, both (ii) and (iii) hold. □

The following proposition follows directly from the lemma just presented.

Proposition 1. *Let α be a weighted regular expression. Then*

$$A_G(\alpha) = \text{rmeps}(A_T(\alpha)). \tag{4}$$

Proof. The only states potentially accessible in $\text{rmeps}(A_T(\overline{\alpha}))$ are the states q_i, $i \geq 0$, since they are the only states with non-ϵ incoming transitions. A state q_i is final with weight w in $\text{rmeps}(A_T(\overline{\alpha}))$ iff $(F_{A_T(\overline{\alpha})}, w) \in \text{closure}(q_i)$, that is, by Lemma 1, iff $(i, w) \in \text{last}_0(\overline{\alpha})$. (q_j, a_i, w, q_i) is a transition in $\text{rmeps}(A_T(\overline{\alpha}))$ iff $(p_i, w) \in \text{closure}(q_j)$, that is, by Lemma 1, iff $(i, w) \in \text{follow}(\overline{\alpha}, j)$. Thus, $A_G(\overline{\alpha}) = \text{rmeps}(A_T(\overline{\alpha}))$. □

This proposition suggests a natural algorithm to compute the Glushkov automaton. The following lemma helps determine its complexity.

Lemma 2. *Let A be the Thompson automaton of a weighted regular expression over a k-closed semiring and let s be a state of A. Then, the shortest-distance algorithm of [19] can be used to compute the shortest distances from the source state s to all states of A in linear time.*

Proof. We give a sketch of the proof. The complexity of the single-source shortest-distance algorithm of [19] depends on the queue discipline used, that is the order in which states are extracted from the queue. One can use a queue discipline that takes advantage of the specific structure of the Thompson automaton. Each subterm of the form $\beta + \gamma$ or β^* defines a sub-automaton with an entry state and an exit state. The appropriate queue discipline enforces that each sub-automaton be fully visited before being exited. The algorithm of [19] can also be modified to store the shortest-distance through the sub-automaton of a β^* subterm once it has been computed and avoid a subsequent revisit. With that queue discipline, the complexity of the algorithm is linear. □

Theorem 1. *Let α be a weighted regular expression over a semiring $\mathbb{K}$ null-k-closed for α and let $m = |\alpha|$ and $n = |\alpha|_\Sigma$. Then, the Glushkov automaton of α can be constructed in time $O(mn)$ by applying ϵ-removal to its Thompson automaton.*

Proof. If $\mathbb{K}$ is null-k-closed for α, then $\mathbb{K}$ is k-closed for all the paths considered during the computation of the ϵ-closures and, by Lemma 2, each ϵ-closure can be computed in linear time, that is in $O(m)$. Since $n + 1$ closures need to be computed, the total complexity is in $O(mn + n^2) = O(mn)$. □

In the unweighted case, the unpublished manuscript of [10] showed that the Glushkov automaton could be obtained by removing the ϵ-transitions from the Thompson automaton using a special-purpose ϵ-removal algorithm.

4 Follow Automaton

The *follow automaton* of an unweighted regular expression α, denoted by $A_F(\alpha)$ was introduced by [12]. Figure 1(c) shows an example. It is the quotient of $A_G(\alpha)$ by the equivalence relation $\equiv_F$ defined over $\text{pos}_0(\alpha)$ by:

$$i \equiv_F j \text{ iff } \begin{cases} \{i,j\} \subseteq \mathrm{last}_0(\overline{\alpha}) \text{ or } \{i,j\} \cap \mathrm{last}_0(\overline{\alpha}) = \emptyset, \text{ and} \\ \mathrm{follow}(\overline{\alpha}, i) = \mathrm{follow}(\overline{\alpha}, j). \end{cases} \tag{5}$$

Proposition 2. *For any regular expression α, the following identities hold:*

$$A_F(\overline{\alpha}) = \min(A_G(\overline{\alpha})) \text{ and } A_F(\alpha) = \overline{\min(A_G(\overline{\alpha}))}.$$

Note that it is mentioned in [12] that minimization could be used to construct the follow automata but the authors claim that the complexity of minimization would be in $O(n^2 \log n)$ making this approach less efficient. The following lemma shows that minimization has in fact a better complexity in this case. Observe that $A_G(\overline{\alpha})$ is deterministic.

Lemma 3. *The time complexity of Hopcroft's minimization algorithm applied to $A_G(\overline{\alpha})$ is linear in the size of $A_G(\overline{\alpha})$: it is in $O(n^2)$ where $n = |\alpha|_\Sigma$.*

Proof. We give a sketch of the proof. The $\log |Q|$ factor in Hopcroft's algorithm corresponds to the number of times the incoming transitions at a given state q are used to split a subset (tentative equivalence class). In $A_G(\overline{\alpha})$, transitions sharing the same label have all the same destination state (the automaton is *1-local*), thus each incoming transition of a state q can only be used to split a subset once. The number of transitions in $A_G(\overline{\alpha})$ is at most n^2. □

The lemma holds in fact for all 1-local automata. This leads to a simple algorithm for constructing the follow automaton of a regular expression α based on:

$$A_F(\alpha) = \overline{\min(\mathrm{rmeps}(A_T(\overline{\alpha})))}. \tag{6}$$

The complexity of this algorithm is in $O(mn)$ which is the same as that of the more complicated and special-purpose algorithms of [12,7]. When the semiring $\mathbb{K}$ is weakly divisible, zero-sum free, and closed, we can define the *follow automaton* of a weighted regular expression α as: $A_F(\alpha) = \overline{\min(A_G(\overline{\alpha}))}$.

Theorem 2. *Let α be a weighted regular expression over $\mathbb{K}$. If $\mathbb{K}$ is k-closed for the Thompson automaton of α, then the follow automaton of α can be computed in $O(mn)$ by applying epsilon-removal followed by weighted minimization to the Thompson automaton of α.*

Proof. The shortest-distance computation required by weight-pushing can be done in $O(m)$ in the case of $A_T(\overline{\alpha})$ and is preserved by ϵ-removal. The weighted automaton $\mathrm{push}(A_G(\overline{\alpha}))$ is 1-local when considered as a finite automaton over pairs (label, weight), thus Lemma 3 can be applied. □

5 Antimirov Automaton

The definition of the Antimirov automaton of a regular expression is based on that of the *partial derivatives of regular expressions*, which are multisets of pairs of the form (w, α) where $w \in \mathbb{K}$ is a weight and α a weighted regular expression over $\mathbb{K}$. For any weight $k \in \mathbb{K}$ and any regular expression β, we define the following operations:

$$k \otimes (w, \alpha) = (k \otimes w, \alpha) \quad (w, \alpha) \otimes k = (w, \alpha k) \quad (w, \alpha) \cdot \beta = (w, \alpha \cdot \beta), \tag{7}$$

which can be naturally extended to multisets of pairs (w, α). By multisets, we mean that $\{(w, \alpha)\} \cup \{(w, \alpha')\} = \{(w, \alpha), (w, \alpha')\}$. The *partial derivative of α with respect to $a \in \Sigma$* is the multiset of pairs (w, α) defined recursively by:

$$\begin{array}{ll}
\partial_a(\epsilon) \;\; = \partial_a(1) = \emptyset & \partial_a(\beta + \gamma) = \partial_a(\beta) \cup \partial_a(\gamma) \\
\partial_a(b) \;\; = \epsilon \text{ if } a = b,\ \emptyset \text{ otherwise} & \partial_a(\beta \cdot \gamma) \;\; = \partial_a(\beta) \cdot \gamma \cup \mathrm{null}(\beta) \otimes \partial_a(\gamma) \\
\partial_a(k\beta) = k \otimes \partial_a(\beta) & \partial_a(\beta^*) \quad\; = \mathrm{null}(\beta)^* \otimes \partial_a(\beta) \cdot \beta^* \\
\partial_a(\beta k) = \partial_a(\beta) \otimes k. &
\end{array}$$

The *partial derivative of α with respect to the string $s \in \Sigma^*$* is denoted by $\partial_s(\alpha)$ and recursively defined by $\partial_{sa}(\alpha) = \partial_a(\partial_s(\alpha))$. Let $D(\alpha) = \{\beta : (w, \beta) \in \partial_s(\alpha)$ with $s \in \Sigma^*$ and $w \in \mathbb{K}\}$. Note that for $D(\alpha)$ to be well-defined, we need to define when two expressions are the same. Here, we will only allow the following identities: $\emptyset \cdot \alpha = \alpha \cdot \emptyset = \emptyset$, $\emptyset + \alpha = \alpha + \emptyset = \emptyset$, $\overline{0}\alpha = \alpha\overline{0} = \emptyset$, $\epsilon \cdot \alpha = \alpha \cdot \epsilon = \alpha$, $\overline{1}\alpha = \alpha\overline{1} = \alpha$, $k(k'\alpha) = (k \otimes k')\alpha$, $(\alpha k)k' = \alpha(k \otimes k')$ and $(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$.[2]

The *Antimirov* or *partial derivatives automaton* $A_A(\alpha)$ of α is then the automaton defined by $A_A(\alpha) = (\Sigma, D(\alpha), E, \alpha, \overline{1}, F, \mathrm{null})$ where $E = \{(\beta, a, w, \gamma)\colon w = \bigoplus_{(w', \gamma) \in \partial_a(\beta)} w'\}$ and $F = \{\beta \in D(\alpha)\colon \mathrm{null}(\beta) \neq \overline{0}\}$. Figure 1(d) shows the Antimirov automaton for a specific regular expression.

Let $\widehat{\Sigma} = \Sigma \cup \{\epsilon^1_+, \epsilon^2_+, \epsilon^1_*, \epsilon^2_*\}$. We denote by $\widehat{A_T(\alpha)}$ the weighted automaton over $\widehat{\Sigma}$ obtained by recursively marking some of the ϵ-transitions of the Thompson automaton $A_T(\alpha)$ as follows: if $\alpha = \beta + \gamma$, we label by ϵ^1_+ (ϵ^2_+) the ϵ-transition from $I_{A_T(\alpha)}$ to $I_{A_T(\beta)}$ (resp. $I_{A_T(\gamma)}$); if $\alpha = \beta^*$, we label by ϵ^1_* (ϵ^2_*) the two ϵ-transitions to $I_{A_T(\beta)}$ (resp. $F_{A_T(\alpha)}$). Observe that $\widehat{A_T(\alpha)}$ can be viewed as an automaton recognizing the expression $\widehat{\alpha}$ over $\widehat{\Sigma}$ recursively defined by $\widehat{\emptyset} = \emptyset$, $\widehat{\epsilon} = \epsilon$, $\widehat{a} = a$, $\widehat{k\beta} = k\widehat{\beta}$, $\widehat{\beta k} = \widehat{\beta}k$, $\widehat{\beta + \gamma} = \epsilon^1_+\widehat{\beta} + \epsilon^2_+\widehat{\gamma}$, $\widehat{\beta \cdot \gamma} = \widehat{\beta} \cdot \widehat{\gamma}$ and $\widehat{\beta^*} = (\epsilon^1_\star\widehat{\beta})^*\epsilon^2_\star$.

For $i \in \mathrm{pos}_0(\alpha)$, we use the same notation q_i (with $q_0 = I$) for the corresponding states in $A_T(\alpha)$, $\widehat{A_T(\alpha)}$ and $\mathrm{rmeps}(\widehat{A_T(\alpha)})$. For a state q in $\mathrm{rmeps}(\widehat{A_T(\alpha)})$, we define by $L(q)$ the language recognized from q considering $\mathrm{rmeps}(\widehat{A_T(\alpha)})$ as an unweighted automaton over pairs (symbol,weight). Lemma 4 follows from our marking of the ϵ-transitions.

Lemma 4. *For $i \in \mathrm{pos}_0(\alpha)$, $L(q_i)$ uniquely defines a regular expression over Σ, denoted by δ_i (or δ_i^α in the presence of ambiguity).*

Lemma 5. *For all $i \in \mathrm{pos}_0(\alpha)$ and $j \in \mathrm{pos}(\alpha)$, we have for p_j, q_i in $A_T(\alpha)$ that:*

$$(p_j, w) \in \mathrm{closure}(q_i) \textit{ iff } (w, \delta_j) \in \partial_a(\delta_i). \tag{8}$$

[2] These identities are the *trivial identities* considered in [15] except for the last two which were added to simplify our presentation. Any larger set of identities can be handled with our method by rewriting α in the corresponding normal form.

Proof. The proof is by induction on the length of the regular expression. If $\alpha = a$, $\alpha = \epsilon$ or $\alpha = \emptyset$, then the properties trivially hold. We give the proof in the case $\alpha = \beta \cdot \gamma$, other cases can be treated similarly.

Let $A = A_T(\alpha)$, $B = A_T(\beta)$ and $C = A_T(\gamma)$. If q_i is in C, then $\delta_i^\alpha = \delta_i^\gamma$ and $\text{closure}_A(q_i) = \text{closure}_C(q_i)$. Therefore, if $(w, p_j) \in \text{closure}_A(q_i)$, p_j is in C and then $\delta_j^\alpha = \delta_j^\gamma$. Hence (8) recursively holds.

If q_i is in B, then $\delta_i^\alpha = \delta_i^\beta \cdot \gamma$ and we have:

$$\partial_a(\delta_i^\alpha) = \partial_a(\delta_i^\beta) \cdot \gamma \cup \text{null}(\delta_i^\beta) \otimes \partial_a(\gamma) \tag{9}$$

$$\text{closure}_A(q_i) = \text{closure}_B(q_i) \cup \text{null}(\delta_i^\beta) \otimes \text{closure}_C(I_C). \tag{10}$$

By induction, we have $(p_j, w) \in \text{closure}_B(q_i)$ iff $(w, \delta_j^\beta) \in \partial_a(\delta_i^\beta)$, and $(p_j, w) \in \text{closure}_C(I_C)$ iff $(w, \delta_j^\gamma) \in \partial_a(\delta_0^\gamma) = \partial_a(\gamma)$. Hence (8) follows. □

Note that $\delta_0 = \alpha$, thus Lemma 5 implies that the δ_i are the derived terms of α, more precisely, $i \mapsto \delta_i$ is a surjection from $\text{pos}_0(\alpha)$ onto $D(\alpha)$. This leads us to the following result, where $\min_{\mathbb{B}}$ is unweighted minimization when each pair (label, weight) is treated as regular symbol and $\widehat{\text{rmeps}}$ denotes the removal of the marked ϵ's.

Proposition 3. *We have* $A_A(\alpha) = \widehat{\text{rmeps}}(\min_{\mathbb{B}}(\text{rmeps}(\widehat{A_T(\alpha)})))$.

Proof. Note that $\text{rmeps}(\widehat{A_T(\alpha)})$ is deterministic. During minimization, two states q_i and q_j are merged iff $L(q_i) = L(q_j)$, that is, by Lemma 4, iff $\delta_i = \delta_j$. Thus, there is a bijection between $D(\alpha)$ and the set of states of $\min_{\mathbb{B}}(\text{rmeps}$ $(\widehat{A_T(\alpha)}))$ having an incoming transition with label in Σ, and thus also between $D(\alpha)$ and the set of states of $A = \widehat{\text{rmeps}}(\min_{\mathbb{B}}(\text{rmeps}(\widehat{A_T(\alpha)})))$. Lemma 5 ensures that the transitions of A are consistent with the definition of $A_A(\alpha)$. □

Theorem 3. *Let α be a weighted regular expression over $\mathbb{K}$. If $\mathbb{K}$ is null-k-closed for α, then the Antimirov automaton of α can be computed in $O(m \log m + mn)$ using ϵ-removal and minimization.*

Theorem 3 follows from the fact that $\text{rmeps}(\widehat{A_T(\alpha)})$ has $O(m)$ states and transitions. In the unweighted case, this complexity matches that of the more complicated and best known algorithm of [8].

In the weighted case, the use of minimization over (label,weight) pairs is suboptimal since states that would be equivalent modulo a $\otimes$-multiplicative factor are not merged. When possible, using weighted minimization instead would lead to a smaller automaton in general. For example, if $\mathbb{K}$ is closed, we can defined the *normalized Antimirov automaton* of α as $\widehat{\text{rmeps}}(\min_{\mathbb{K}}(\text{rmeps}(\widehat{A_T(\alpha)})))$. This automaton is always smaller than the Antimirov automaton and the automaton of unitary derived terms of [15].[3] When $\mathbb{K}$ is k-closed, it can be constructed in $O(m \log m + mn)$.

[3] This automaton can be viewed in our approach as the result of a simpler form of reweighting than weight-pushing, the reweighting used by weighted minimization.

Remark. When the condition about k-closedness (null-k-closedness for α) of $\mathbb{K}$ is relaxed to the closedness of $\mathbb{K}$ (resp. that α is well-defined), all our construction algorithms can still be used by replacing the generic single-source shortest-distance algorithm with a generalization of the Floyd-Warshallalgorithm [14,19], leading to a complexity of $O(m^3)$. It is not hard however to maintain the quadratic complexity by modifying the generic single-source shortest-distance algorithm to take advantage of the special topology of the Thompson automaton.

In the unweighted case, every regular expression can be straightforwardly rewritten in ϵ-normal form such that $m = O(n)$. In that case, our $O(mn)$ and $O(m \log m + mn)$ complexities become $O(m + n^2)$ which coincides with what is often reported in the literature.

6 Conclusion

We presented a simple and unified view of ϵ-free automata representing unweighted and weighted regular expressions. We showed that standard unweighted and weighted epsilon-removal and minimization algorithms can be used to create the Glushkov, follow, and Antimirov automata and that the time complexity of our construction algorithms is at least as favorable as that of the best previously known algorithm.

This provides a better understanding of the ϵ-free automata representing regular expressions. It also suggests using other combinations of epsilon-removal and minimization for creating ϵ-free automata. For example, in some contexts, it might be beneficial to use reverse-epsilon-removal rather than epsilon-removal [18]. Note also that the Glushkov automaton can be constructed on-the-fly since Thompson's construction and epsilon-removal both admit an on-demand implementation.

Acknowledgments. This project was sponsored in part by the Department of the Army Award Number W23RYX-3275-N605. The U.S. Army Medical Research Acquisition Activity, 820 Chandler Street, Fort Detrick MD 21702-5014 is the awarding and administering acquisition office. The content of this material does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred. This work was also partially funded by the New York State Office of Science Technology and Academic Research (NYSTAR).

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques and Tools*. Addison Wesley: Reading, MA, 1986.
2. V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.
3. G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48(3):117–126, 1986.
4. A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.

5. P. Caron and M. Flouret. Glushkov construction for series: the non commutative case. *International Journal of Computer Mathematics*, 80(4):457–472, 2003.
6. J.-M. Champarnaud, É. Laugerotte, F. Ouardi, and D. Ziadi. From regular weighted expressions to finite automata. In *Proceedings of CIAA 2003*, volume 2759 of *Lecture Notes in Computer Science*, pages 49–60. Springer-Verlag, 2003.
7. J.-M. Champarnaud, F. Nicart, and D. Ziadi. Computing the follow automaton of an expression. In *Proceedings of CIAA 2004*, volume 3317 of *Lecture Notes in Computer Science*, pages 90–101. Springer-Verlag, 2005.
8. J.-M. Champarnaud and D. Ziadi. Computing the equation automaton of a regular expression in $O(s^2)$ space and time. In *Proceedings of CPM 2001*, volume 2089 of *Lecture Notes in Computer Science*, pages 157–168. Springer-Verlag, 2001.
9. C.-H. Chang and R. Page. From regular expressions to DFA's using compressed NFA's. *Theoretical Computer Science*, 178(1-2):1–36, 1997.
10. D. Giammarresi, J.-L. Ponty, and D. Wood. Glushkov and Thompson constructions: a synthesis. `http://www.cs.ust.hk/tcsc/RR/1998-11.ps.gz`, 1998.
11. V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.
12. L. Ilie and S. Yu. Follow automata. *Information and Computation*, 186(1):146–162, 2003.
13. S. C. Kleene. Representations of events in nerve sets and finite automata. In C. E. Shannon, J. McCarthy, and W. R. Ashby, editors, *Automata Studies*, pages 3–42. Princeton University Press, 1956.
14. D. J. Lehmann. Algebraic structures for transitives closures. *Theoretical Computer Science*, 4:59–76, 1977.
15. S. Lombardy and J. Sakarovitch. Derivatives of rational expressions with multiplicity. *Theoretical Computer Science*, 332(1-3):142–177, 2005.
16. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9(1):39–47, 1960.
17. M. Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:2, 1997.
18. M. Mohri. Generic e-removal and input e-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.
19. M. Mohri. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
20. G. Navarro and M. Raffinot. Fast regular expression search. In *Proceedings of WAE'99*, volume 1668 of *Lecture Notes in Computer Science*, pages 198–212. Springer-Verlag, 1999.
21. G. Navarro and M. Raffinot. *Flexible pattern matching.* Cambridge University Press, 2002.
22. J.-L. Ponty, D. Ziadi, and J.-M. Champarnaud. A new quadratic algorithm to convert a regular expression into automata. In *Proceedings of WIA'96*, volume 1260 of *Lecture Notes in Computer Science*, pages 109–119. Springer-Verlag, 1997.
23. M.-P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
24. K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):365–375, 1968.

Algebraic Characterizations of Unitary Linear Quantum Cellular Automata

Pablo Arrighi

IMAG Laboratories & University of Grenoble,
46 Avenue Félix Viallet, 38031 Grenoble Cedex, France
pablo.arrighi@imag.fr

Abstract. We provide algebraic criteria for the unitarity of linear quantum cellular automata, i.e. one dimensional quantum cellular automata. We derive these both by direct combinatorial arguments, and by adding constraints into the model which do not change the quantum cellular automata's computational power. The configurations we consider have finite but unbounded size.

1 Motivations

One could say that the central question in theoretical computer science is 'What are the resources necessary for computation, or information processing?'. Ultimately this question is dictated by the physical laws which surround us. Quantum computation science has risen from this basic idea. It considers computers as physical, and hence possibly quantum systems. At the theoretical level it was demonstrated for instance that polynomial-time integer factorization is possible with such systems, as well as the search for an element in a unordered list of size n in time $\Theta(\sqrt{n})$.

Cellular automata (CA) are arrays of cells, each of which may take one in a finite number of possible state. These evolve in discrete time steps according to a global evolution Δ – which itself arises from the application of a local transition function δ, synchronously and homogeneously across space. A popular example is Conway's 'Game of Life', a two dimensional CA which has been proven to be universal for computation.

It is clear that CA are themselves physics-like models of computations, in the sense that they describe a world of small systems interacting locally, according to translation-invariant laws. Therefore it seems natural to study their quantum extensions. Moreover no classical control is required in such models, since computation arises as an emergent behaviour of the quantum cells' interaction. This is a key advantage to have as it reduces the need for environment interaction, and hence may reduce decoherence, the principal obstacle to realizing a quantum computer. For these two reasons Feynman [6], in his seminal paper about quantum computation, has argued that the study of quantum CA may prove an important path to a realistic physical implementation of quantum computers. Another series of legitimate aims is to endow quantum computation with spatio-temporal notions, or even to provide a bridge through which computer science notions, such as universality, may contribute to modern theoretical physics. To put it differently such works are a contribution to the understanding of dynamics in discrete, quantum spacetime – but from an idealized, computer-science viewpoint.

Similarly to their classical counterparts linear quantum cellular automata (LQCA) consist of a row of identical, finite dimensional, quantum systems. These evolve in discrete time steps according to a global evolution Δ – which itself arises from the application of a local transition function δ, homogeneously and synchronously across space. But in order to grant LQCA the status of physically acceptable model of computation, one must ensure that the global evolution Δ is physically acceptable in a quantum theoretical setting, i.e. one must ensure that Δ is unitary. Unfortunately this global property is rather non-trivially related to the description of the local transition function δ – witness of this the abundant literature on reversible cellular automata (RCA), tackling the classical counterpart of this issue. It is actually a very surprising fact that so much has been done to study RCA – when reversibility is not so much of a crucial feature to have in classical computation. A frequently encountered argument states that all consumption-less, zero-heat micro-mechanical device need be reversible. But tracing back the origins of this argument, we find quantum physical considerations once more [9].

One way to approach this issue is to find a decision procedure, which given δ tells whether Δ is unitary. This test should be performed efficiently, so as not to carry any of the complexity of the computation, and be applied to any candidate δ as a mean to exclude the non-physical ones. Such a strong contribution was indeed achieved by Dürr et al. It does not put an ending point to the problem however, because the number of local transition functions which do indeed induce a unitary global evolution is likely to be rather scarce, as was the case for RCA [1]. Moreover this relatively complicated decision procedure takes an elegant detour via finite automata, but one which does not guide us to understand which δ's will eventually yield a physical Δ.

Physicists are used to checking whether an evolution is physically acceptable, but they like to do so algebraically (e.g. $U^\dagger U = UU^\dagger = \mathbb{I}$ for unitarity, when U is a finite matrix). Much in the same way computer scientists are used to checking whether a program is valid, but like these criteria to be syntactic. When this is not the case, we tend to consider that the definition chosen to formalize the model of computation is in fact too loose. Indeed once universality has been reached, adding more expressiveness does not mean adding more computational power, but only more ways of expressing the same computation. An undesirable excess arises when the syntax proposed by the definition allows the description of non-valid programs, thereby requiring that the user performs elaborate decision procedures to exclude those instances. This is the current state of affairs with LQCA.

Therefore a different, non-complexity-theoretic approach is to tighten the definition of linear quantum cellular automata, i.e. to seek for a more restrictive definition whose unitarity may be checked algebraically/syntactically, and yet capable of expressing rigorously the same set of global evolutions as our original definition. In this paper we provide some synthetic algebraic formulae which characterize unitary linear quantum cellular automata. We are not interested in the complexity of running the corresponding computations, and in particular we do not deal with the difficulties associated to computing with complex number up to arbitrary precision and/or the approximations which finite precision arithmetic entails. Our criteria are derived both by direct combinatorial arguments, and by adding constraints into the model which do not change the quantum cellular automata's computational power.

The breakdown of this paper will be given after LQCA are presented formally, in the following section. Our main theorem is stated in the conclusion section, and discussed in comparison to some related approaches.

Notations. Throughout the paper we will denote by $\mathcal{H}_S$ the Hilbert space whose canonical orthonormal base vectors are identified with the elements of the countable set S. E.g. $\mathcal{H}_{\{aa,ab,ba,bb\}}$ is the four dimensional space with canonical orthonormal base $\{|aa\rangle, |ab\rangle, |ba\rangle, |bb\rangle\}$. This means that any vector $\alpha|aa\rangle + \beta|ab\rangle + \gamma|ba\rangle + \delta|bb\rangle$ with $\alpha, \beta, \gamma, \delta \in \mathbb{C}$ belongs to $\mathcal{H}_{\{aa,ab,ba,bb\}}$. Such a vector must be thought of as a superposition of the words aa, ab, ba, bb. Moreover the symbol $\mathbf{0}$ is to denote the null vector, not to be confused with O the matrix containing only ones.

2 The Model

We start with the definition proposed by [15] and [4], sometimes also referred to as linear quantum cellular automata (LQCA). This definition will evolve throughout the paper.

Working definition 1 (LQCA).
A linear quantum cellular automaton (LQCA) is a 4-tuple $\mathcal{A} = (\Sigma, q, N, \delta)$, where (with $q\Sigma = \{q\} \cup \Sigma$):

- Σ is a finite set of symbols (i.e. "the alphabet", giving the possible basic states each cell may take);
- q is a symbol such that $q \notin \Sigma$ (i.e. "the quiescent symbol", which may be thought as a special state for empty cells);
- N is a set of n successive signed integers (i.e. "the neighbourhood", telling which cell is next to whom);
- $\delta : \mathcal{H}_{(q\Sigma)^n} \longrightarrow \mathcal{H}_{q\Sigma}$ is a function from superpositions of n symbols words to superpositions of one symbol words (i.e. "the local transition function", describing the way a cell interacts with its neighbours).

Moreover δ must verify:

- the quiescent stability condition: $\big[\delta|q^n\rangle) = |q\rangle\big]$;
- the no-nullity condition: $\forall w \in (q\Sigma)^n$, $\big[\delta|w\rangle \neq \mathbf{0}\big]$.

By 'successive' we mean that the number follow each other in unit step, i.e. the neighbourhood is an interval. In the literature these are sometimes referred to as simple neighbourhoods, but it is trivial to simulate non-simple neighbourhoods automata with simple neighbourhoods automata. At this point we need not have a normalization condition such as $\forall w \in (q\Sigma)^n,\ \big[||\delta|w\rangle|| = 1\big]$. Configurations hold the basic states of an entire row of cells. As we will now formalize ours are finite but unbounded. Note that fixed-sized periodic configurations[14] as well as infinite configurations[13] have also been studied, leading to very different results and proof methods (see Sect. 6 for a discussion).

Definition 1 (Finite configurations, interval domains).
A (finite) configuration *c of the quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$ is a function $c : \mathbb{Z} \longrightarrow q\Sigma$, with $i \longmapsto c(i) = c_i$, such that there exists a (possibly empty) interval I verifying $i \in I \Rightarrow c_i \in q\Sigma$ and $i \notin I \Rightarrow c_i = q$. Finally the set of all finite configurations is denoted $\mathcal{C}_f$.*

Definition 2 (Indexing conventions).
Given a configuration c of the quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$, we denote by $c_{k\ldots l}$ the word $c_k \cdot \ldots \cdot c_l$ if $k \leq l$, and the empty word ε otherwise. Thus in either case $c_{k\ldots l} \in (q\Sigma)^*$. Moreover we denote by c_{i+N} the word $c_{i+\min(N)\ldots i+\max(N)}$, and $c_{i+\tilde{N}}$ the word $c_{i+\min(N)\ldots i+\max(N)-1}$. Therefore we have $c_{i+N} \in (q\Sigma)^n$ and $c_{i+\tilde{N}} \in (q\Sigma)^{n-1}$, respectively.

Whilst configurations hold the basic states of an entire row of cells, and hence denote the possible basic states of the entire LQCA, the global state of a LQCA may well turn out to be a superposition of these. The following definition works because $\mathcal{C}_f$ is a countably infinite set.

Definition 3 (Superpositions of finite configurations).
A superposition of configurations *of the quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$ is a normalized element of $\mathcal{H}_{\mathcal{C}_f}$, the Hilbert space of configurations.*

Definition 4 (Global evolution).
The global evolution *of the quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$ is the linear operation defined by linear extension of its action upon the canonical orthonormal basis, as follows:*

$$\begin{aligned} \Delta : \mathcal{H}_{\mathcal{C}_f} &\to \mathcal{H}_{\mathcal{C}_f} \\ |c\rangle &\mapsto \Delta|c\rangle \\ \Delta|c\rangle &= \bigotimes_{i\in\mathbb{Z}} \delta|c_{i+N}\rangle \end{aligned}$$

The postulates of quantum theory impose that the global evolution should be unitary.

Definition 5 (Unitarity).
The global evolution *Δ of the quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$ is said to be* (finite) unitary *if and only if $\{\Delta|c\rangle \mid c \in \mathcal{C}_f\}$ is an orthonormal basis of $\mathcal{H}_{\mathcal{C}_f}$.*

The next lemma recalls some facts of the linear algebra which is commonly used in quantum information theory [11].

Lemma 1 (Norm-preservedness, unitarity).
Let $\Delta : \mathcal{H}_S \rightarrow \mathcal{H}_S$ be a linear operator over a Hilbert space $\mathcal{H}_S$ having canonical orthonormal basis $\{|c\rangle\}_{c\in S}$. Suppose the following two conditions are fulfilled simultaneously:

-(i) $\forall c \in S$, $[||\Delta|c\rangle|| = 1]$;
-(ii) $\forall c, c' \in S$, $[\langle c'|\Delta^\dagger\Delta|c\rangle \neq 0 \Leftrightarrow c = c']$.
Then $\forall r \in S$, $[0 \leq ||\Delta^\dagger|r\rangle|| \leq 1]$.
Moreover if
-(iii) $\forall r \in S$, $[||\Delta^\dagger|r\rangle|| = 1]$.
Then Δ is unitary.

Proof. Conditions (i) and (ii) express the fact that Δ is norm-preserving. As a consequence for all $r \in S$ we have $||\Delta^\dagger|r\rangle|| = ||\Delta\Delta^\dagger|r\rangle||$. Moreover we have by definition $||\Delta^\dagger|r\rangle||^2 = |\langle r|\Delta\Delta^\dagger|r\rangle|$. But the latter ($\langle r|\Delta\Delta^\dagger|r\rangle$) is a projection of the former ($\Delta\Delta^\dagger|r\rangle$) over a unit vector ($|r\rangle$), and hence $|\langle r|\Delta\Delta^\dagger|r\rangle| \leq ||\Delta\Delta^\dagger|r\rangle||$. Therefore $||\Delta^\dagger|r\rangle||^2 \leq ||\Delta^\dagger|r\rangle||$ and so $||\Delta^\dagger|r\rangle|| \leq 1$.

Condition (iii) expresses the fact that for all $r \in S$, $\Delta^\dagger|r\rangle$ has unit norm. As a consequence $\Delta\Delta^\dagger|r\rangle$ has unit norm on the one hand, and $\langle r|\Delta\Delta^\dagger|r\rangle = ||\Delta^\dagger|r\rangle||^2 = 1$ on the other hand. Therefore $\alpha\Delta\Delta^\dagger|r\rangle = |r\rangle$, with α a root of unity. Since $\Delta\Delta^\dagger$ is positive, this α is just 1. Let $\Delta^\dagger|r\rangle = \sum \beta_c|c\rangle$. Then $|r\rangle = \sum \beta_c\Delta|c\rangle$, in other words each of the canonical orthonormal basis vectors $|r\rangle$ may be expressed as a linear combination of columns $\{\Delta|c\rangle \mid c \in S\}$. Therefore the columns form themselves an orthonormal basis. □

Note for later use that if (i) and (ii) are fulfilled and T is a finite subset of S, then $[\sum_{r\in T} ||\Delta^\dagger|r\rangle||^2 = |T|]$ implies $\forall r \in T$, $[||\Delta^\dagger|r\rangle|| = 1]$. Next we will examine each of the following conditions in turn:

-(i) the columns of Δ have unit norm (Sect. 3);
-(ii) the columns of Δ are orthogonal (Sect. 4);
-(iii) the rows of Δ have unit norm (Sect. 5).

3 Unit Columns

The next two lemmas are simple facts from [4].

Lemma 2 (Norm of a column).
Let Δ denote the global evolution *of the quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$. We have that*

$$\forall c \in \mathcal{C}_f, \quad [||\Delta|c\rangle|| = \prod_{i\in\mathbb{Z}} ||\delta|c_{i+N}\rangle||]$$

Proof. The norm of a tensor product of vectors is the product of the norms of the vectors. □

Lemma 3 (Expressiveness of normalized δ).
Let Δ denote the global evolution of the quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$. Let Δ' denote the global evolution of the quantum cellular automaton $\mathcal{A}' = (\Sigma, q, N, \delta')$, with δ' such that $\forall w \in (q\Sigma)^n$, $\big[\delta'|w\rangle = \delta|w\rangle / ||\delta|w\rangle||\big]$. Suppose that the columns of Δ have unit norm. Then we have $\Delta = \Delta'$.

Proof. For all c in $\mathcal{C}_f$ we have

$$\begin{aligned}\Delta|c\rangle &= \bigotimes_{i\in\mathbb{Z}} \delta|c_{i+N}\rangle = \bigotimes_{i\in\mathbb{Z}} ||\delta|c_{i+N}\rangle||.\delta'|c_{i+N}\rangle \\ &= \prod_{i\in\mathbb{Z}} ||\delta|c_{i+N}\rangle||.\bigotimes_{i\in\mathbb{Z}} \delta'|c_{i+N}\rangle = ||\Delta|c\rangle||.\bigotimes_{i\in\mathbb{Z}} \delta'|c_{i+N}\rangle = \Delta'|c\rangle \qquad \square\end{aligned}$$

Our approach is to change the actual definition of LQCA as a consequence.

Working definition 2 (LQCA).
As in Working Def. 1 but remove the no-nullity condition and add the normalization condition:

$\forall w \in (q\Sigma)^n$, $\big[||\delta|w\rangle|| = 1\big]$.

This modified definition of LQCA choses to impose that $\forall w \in (q\Sigma)^n$, $\big[||\delta|w\rangle|| = 1\big]$, i.e. that the local transition function δ is normalized. Then the fact that the columns of Δ have unit norm follows straight from Lem. 2. Note that we can omit the no-nullity condition as it is implied by the normalization condition.

Surely this is not the only way to tackle the problem, but this one has been chosen due to the many advantage arising:

- The alternative is to have various non-normalized states compensating each other non-locally, which from a physical point of view is somewhat disturbing;
- It saves us from having to employ more elaborate techniques to check that columns have unit norms[4][7], such as applying least path algorithm to the associated de Bruijn graphs of the quantum cellular automata etc. Although very elegant these tend to render quantum cellular automata much more oblivious as a model of computation;
- The modification made has absolutely no cost in terms expressiveness, as demonstrated in Lemma 3;
- The normalization condition will later bring out more crucial simplifications.

4 Orthogonality of Columns

Having checked that the columns of the global evolution matrix Δ have unit norm, we now turn to the problem of deciding whether these columns are mutually orthogonal. First we need a definition.

Definition 6 (A-matrix).
Consider a linear quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$. We call $A = \big[A_{xy}\big]$ with $x, y \in (q\Sigma)^{n-1}$ the matrix (tensor) such that $A_{xy} = \sum_{\sigma\in q\Sigma} A^{\sigma}_{xy}|\sigma\rangle$ equals $\delta|w\rangle$ if we have both $x = w_{1\ldots n-1}$ and $y = w_{2\ldots n}$ for some $w \in (q\Sigma)^n$, otherwise it is the null vector.

For instance consider the sample rule: $\delta|000\rangle = |0\rangle$, $\delta|001\rangle = |1\rangle$, $\delta|010\rangle = |1\rangle$, $\delta|011\rangle = |0\rangle$, $\delta|100\rangle = |0\rangle$, $\delta|101\rangle = |1\rangle$, $\delta|110\rangle = |1\rangle$, $\delta|111\rangle = |0\rangle$.

Then the A-matrix of the sample rule is: $\begin{pmatrix} |0\rangle & \mathbf{0} & |0\rangle & \mathbf{0} \\ |1\rangle & \mathbf{0} & |1\rangle & \mathbf{0} \\ \mathbf{0} & |1\rangle & \mathbf{0} & |1\rangle \\ \mathbf{0} & |0\rangle & \mathbf{0} & |0\rangle \end{pmatrix}$. E.g. since 01 and 00 do not "follow each other" the entry $\langle 00|A|01\rangle$ holds the null vector. On the other hand 01 and 10 do overlap correctly to form the neighbourhood 010, for which $\delta|010\rangle = |1\rangle$, hence $\langle 10|A|01\rangle = |1\rangle$.

How can we check that *all* columns of some global evolution are mutually orthogonal, when there is infinitely many of them? Our next proposition is crucial in that respect, as it shows why the problem which might seem to be of an infinite (undecidable) nature is indeed of a finite (decidable) nature.

Proposition 1 (Finite columns checks).
This result refers to working definition 2.

Consider the global evolution Δ of a quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$. Let s be equal to $|q\Sigma|^{2n-2} - 1$ and I be the interval $[0, s]$. The columns $\{\Delta|c\rangle \mid c \in \mathcal{C}_f\}$ are orthogonal if and only if the columns $\{\Delta|c\rangle \mid c \in \mathcal{C}_f^I\}$ are orthogonal.

(See [2] for a detailed proof.)
We now give our algebraic condition upon δ ensuring that the columns of Δ are mutually orthogonal.

Proposition 2 (Column test).
This result refers to working definition 2.

Consider the global evolution Δ of a quantum cellular automaton $\mathcal{A} = (\Sigma, q, N, \delta)$. Let $s = |q\Sigma|^{2n-2} - 1$. The columns $\{\Delta|c\rangle \mid c \in \mathcal{C}_f\}$ are orthogonal if and only if $\forall x, x' \in (q\Sigma)^{n-1}$:

$$\left[\langle xx'|M^s|q^{n-1}q^{n-1}\rangle\langle q^{n-1}q^{n-1}|M^s|xx'\rangle \neq 0 \Leftrightarrow (x = x')\right] \tag{1}$$

with $M = [M_{xx',yy'}]$, $M_{xx',yy'} = \left|\sum_\sigma A^{\sigma *}_{x'y'} A^{\sigma}_{xy}\right|^2$, A the A-tensor of the LQCA.

(See [2] for a detailed proof.)
We believe that the obtention of algebraic conditions constituted a necessary step in order to be able to take further the analysis of this model. The algebraic proofs of these conditions give them a physical meaning which shortcuts the graph-theoretical detour, which is particularly useful since quantum theory people tend to reason in terms of linear algebra rather than graph theory. They also master the corresponding numerical tools better, Proposition 2 makes it easy to check for column orthonormality through any software tool which does matrix multiplication.

Still there remains some space for improvement, for instance because Proposition 2 is phrased in terms of the somewhat bizarre A-tensor of $\mathcal{A}$, rather than just δ. Fortunately this first point can be fixed using the quantum equivalent of a simplifying classical result[8][12][3]:

Lemma 4 (Expressivity of size two δ).
This result refers to working definition 2.

Consider a linear quantum cellular automaton $\mathcal{A}' = (\Sigma, q, N, \delta')$ and its global evolution Δ'. One can always construct a linear quantum cellular automaton $\mathcal{A} = (\Sigma^{n-1}, q^{n-1}, \{0,1\}, \delta)$ such that its global evolution Δ equals Δ'. Then the A-tensor of $\mathcal{A}$ is just $\delta = \left[\delta^{\sigma}_{xy}\right]$, with $x, y, \sigma \in \Sigma^{n-1}$.

Proof. We let, for all $x, y \in \Sigma^{n-1}$

$$\delta|x_1 \dots x_{n-1} y_1 \dots y_{n-1}\rangle \equiv \bigotimes_{i \in [1,n-1]} \delta'|x_i \dots x_{n-1} y_1 \dots y_i\rangle \ .$$

The rest follows from the definitions. □
Note that the groups of $n-1$ cells constructed in this lemma bear some resemblance with the reduced neighbourhoods $\tilde{N}$ constructed by Dürr et al. [4] – except they do not overlap, which consequently saves us from using De Bruijn graphs and the like. This suggests that in spite of its apparent simplicity this trick is probably just the right way to enumerate/construct LQCA.

We change the definition of LQCA as a consequence.

Working definition 3 (LQCA).
A linear quantum cellular automaton (LQCA) is a 3-tuple $\mathcal{A} = (\Sigma, q, \delta)$, where:

- Σ is a finite set of symbols ("the alphabet");
- q is a symbol such that $q \notin \Sigma$ ("the quiescent symbol");
- $\delta : \mathcal{H}_{(q\Sigma)^2} \rightarrow \mathcal{H}_{q\Sigma}$ is a function from superpositions of 2 symbols words to superpositions of one symbol words ("the local transition function").

Moreover δ must verify the following two properties:

- the quiescent stability condition: $\left[\delta|qq\rangle) = |q\rangle\right]$.
- the normalization condition:

$\forall w \in (q\Sigma)^2,\ \left[||\delta|w\rangle|| = 1\right]$.

This modified definition of LQCA choses to impose neighbourhoods of size two on top of the normalization condition. For these linear quantum cellular automata we define Δ as usual with $N = \{0, 1\}$. Again we are strongly justified to place this easy restriction straight into the definition of LQCA for it simplifies and makes more intuitive the decision procedure induced by Prop. 2, (the corresponding simplifications are shown in Corollary 1). The modification made comes at absolutely no cost in terms expressiveness, as demonstrated in Lemma 4.

These successive two modifications we have made to our model will be even more asserted by the important simplification they bring to the problem of determining whether a LQCA has unit rows.

5 Unit Rows

Having checked that the columns of the global evolution matrix Δ are orthonormal, we now turn to the problem of deciding whether its rows have unit norm.

Proposition 3 (Row norm as matrix product).
This result refers to working definition 3.

Consider a quantum cellular automaton $\mathcal{A} = (\Sigma, q, \delta)$ whose global evolution Δ has orthonormal columns. The squared norm of any row r is given by

$$||\Delta^\dagger|r\rangle||^2 = \lim_{h\to\infty} \langle q|{N^{(q)}}^h \Big(\prod_{i\in k\ldots l} N^{(r_i)}\Big){N^{(q)}}^h|q\rangle$$

where $\{N^{(\sigma)}\}_{\sigma\in q\Sigma}$ is the set of matrices such that $N^{(\sigma)} = [N^{(\sigma)}_{x,y}], N^{(\sigma)}_{x,y} = |\langle\sigma|\delta|xy\rangle|^2$.

(See [2] for a detailed proof.)
The following proposition takes advantage of the successive restrictions made in definitions 2 and 3 to bring about a crucial simplification – which makes obsolete a good half of the procedure described in [5].

Proposition 4 (Middle segment).
This result refers to working definition 3.

Consider a quantum cellular automaton $\mathcal{A} = (\Sigma, q, \delta)$ whose global evolution Δ has orthonormal columns. The rows $\{\Delta^\dagger|r\rangle \mid r \in \mathcal{C}_f\}$ have unit norm if and only if

$$\lim_{h\to\infty} \langle q|N^h O N^h|q\rangle = |q\Sigma|$$

with $O = [1_{xy}]$ the matrix with only ones, and $N = [N_{x,y}], N_{x,y} = |\langle q|\delta|xy\rangle|^2$.

(See [2] for a detailed proof.)
Note that $\lim_{h\to\infty}\langle q|N^h O N^h|q\rangle = \sum_{r\in\mathcal{C}_f^{[0,0]}} ||\Delta^\dagger|r\rangle||^2$. Hence we have the following insightful corollary, which comes as the direct analogue of our Prop. 1. Curiously however it will not contribute to our final result.

Proposition 5 (Finite rows check).
This result refers to working definition 3.

Consider a quantum cellular automaton $\mathcal{A} = (\Sigma, q, \delta)$ whose global evolution Δ has orthonormal columns. The rows $\{\Delta^\dagger|r\rangle \mid r \in \mathcal{C}_f\}$ have unit norm if and only if the rows $\{\Delta^\dagger|r\rangle \mid r \in \mathcal{C}_f^{[0,0]}\}$ have unit norm.

Both results deepen our understanding of the algebraic structure of unitary linear quantum cellular automata. Prop. 4 offers an important simplification along the way to determining whether the global evolution Δ has unit rows, reducing this to the evaluation of $\overrightarrow{l}.O\overrightarrow{r}$.

The difficult problem we are left with is that of evaluating the so-called 'border vectors' [5] $\overrightarrow{r} = \lim_{h\to\infty} N^h|q\rangle$ and $\overrightarrow{l} = \lim_{h\to\infty} {N^\dagger}^h|q\rangle$. The following proposition will characterize them uniquely and algebraically. We let $\leq$ denotes the following partial order upon $m \times n$ matrices:

$$M \leq N \Longleftrightarrow \forall i, j \quad M_{ij} \leq N_{ij} \enspace .$$

Column vectors are seen as $m \times 1$ matrices for that matter.

Proposition 6 (Border vectors).
This result refers to working definition 3.

Consider a quantum cellular automaton $\mathcal{A} = (\Sigma, q, \delta)$ whose global evolution Δ has orthonormal columns. The vectors

$$\overrightarrow{r} = \lim_{h\to\infty} N^h|q\rangle \quad \textit{and} \quad \overrightarrow{l} = \lim_{h\to\infty} {N^\dagger}^h|q\rangle$$

with $N = [N_{x,y}], N_{x,y} = |\langle q|\delta|xy\rangle|^2$, have only finite entries. They verify

$$\overrightarrow{r} = \min_{\leq}\{v \mid \mathbf{0} \leq v \ \wedge \ Nv = v \ \wedge \ v_q = 1\}$$
$$\textit{and} \quad \overrightarrow{l} = \min_{\leq}\{v \mid \mathbf{0} \leq v \ \wedge \ vN = v \ \wedge \ v_q = 1\} \ .$$

Moreover the following extra conditions hold:

(i) $\overrightarrow{l}.\overrightarrow{r} = 1$;
(ii) $(\sum_i \overrightarrow{l}_i)(\sum_i \overrightarrow{r}_i) \leq |q\Sigma|$.
(iii) $\forall x \in \Sigma,\ [\overrightarrow{l}_x = 0 \ \vee \ \overrightarrow{r}_x = 0]$;
(iv) $\forall x, y \in \Sigma,\ [N_{xy} \neq 0 \Rightarrow \overrightarrow{l}_x = 0 \ \vee \ \overrightarrow{l}_y = 0]$;
Inequality (ii) is saturated if and only if Δ has unit rows.

(See [2] for a detailed proof.)
This last proposition is highly informative, and in *most cases* will provide us with an effective way to compute these border vectors, through a spectral decomposition of N. When the eigenvalue 1 is degenerate, however, we leave it as an open problem whether there exists a definite procedure to performing the minimization.

6 Conclusion

The following theorem is a synthesis our results.

Theorem 1 (Summary).
This result refers to working definition 3.

Consider the global evolution Δ of a quantum cellular automaton $\mathcal{A} = (\Sigma, q, \delta)$. Let $s = |q\Sigma|^2 - 1$. Δ is unitary if and only if $\forall x, x' \in (q\Sigma)$:

$$[\langle xx'|M^s|qq\rangle\langle qq|M^s|xx'\rangle \neq 0 \Leftrightarrow (x = x')]$$
$$\textit{and} \quad (\sum_i \overrightarrow{l}_i)(\sum_i \overrightarrow{r}_i) = |q\Sigma|$$
$$\textit{with}\ \overrightarrow{r} = \min_{\leq}\{v \mid \mathbf{0} \leq v \ \wedge \ Nv = v \ \wedge \ v_q = 1\};$$
$$\overrightarrow{l} = \min_{\leq}\{v \mid \mathbf{0} \leq v \ \wedge \ vN = v \ \wedge \ v_q = 1\};$$
$$M = [M_{xx',yy'}],\ M_{xx',yy'} = \left|\langle x'y'|\delta^\dagger\delta|xy\rangle\right|^2;$$
$$N = [N_{x,y}],\ N_{x,y} = |\langle q|\delta|xy\rangle|^2.$$

Once again we stress that our approach is not a complexity-theoretic one. We have not sought to compare ourselves to [4,5] on such a ground, and nor have we sought to evaluate the hidden cost of computations with complex numbers of arbitrary precision, or the approximations any alternative may entail. Our approach is an algebraic one, and in this respect we have definitely gone a long way towards the simplification and the algebraization of unitarity criteria for linear quantum cellular automata as defined in [15], [4], [5]. Note that these last two papers do not contain any such synthetic algebraic criteria. Instead they provide several pages long decision procedures, which have many twists and bends.

Certainly we have not gone as far as to reduce LQCA to partitioned quantum cellular automata (PQCA) or quantum cellular automata with Margolus neighbourhood (MQCA) etc., i.e. our global evolution cannot, in general, be decomposed into the application of one small unitary operator homogeneously across space.

Other models of quantum cellular automata do admit such reductions; the problem of deciding unitarity becomes incomparably easier, or even trivial by construction [13]. Such crucial differences seem to arise when one considers different spaces of configurations, e.g. finite periodic [14], because these constrain reversibility to be structural, i.e. an essentially local matter. With the space of finite configurations $\mathcal{C}_f$ reversibility becomes a global matter, even though the global evolution is defined locally. Analogues of this are well-known in the classical realm already:

Remark 1. Consider Xor' $= (\{q, 0, 1\}, q, \delta)$ with $\delta|00\rangle = |0\rangle$, $\delta|01\rangle = |1\rangle$, $\delta|10\rangle = |1\rangle$, $\delta|11\rangle = |0\rangle$, $\delta|q0\rangle = |q\rangle$, $\delta|q1\rangle = |q\rangle$, $\delta|0q\rangle = |0\rangle$, $\delta|1q\rangle = |1\rangle$. Its corresponding global evolution Δ is bijective in the space of finite configurations (i.e. it is unitary), but the global evolution cannot be reversed by a cellular automata. Moreover Δ is not reversible in the space of infinite configurations ($\ldots 00 \ldots$ may have antecedents either $\ldots 00 \ldots$ or $\ldots 11 \ldots$).

One may argue that PQCA models are enough, since they simulate the quantum Turing machine. If we are interested in 'intrinsic universality' however, and want to consider the above example as 'physical' then these are not enough. A local transition δ which is unitary in the space of infinite configurations is also unitary in the space if finite configurations, but the reciprocal statement is not true. In this sense there may be a loss of expressiveness when restricting to PQCA models.

Note that this research constitutes a first step in the quest for the identification of the one particular unitary LQCA known to simulate all other LQCA efficiently. Obviously as we restrict the number of linear quantum cellular automata to be considered to a well-behaved subclass, the quest for such an 'intrinsically universal' one dimensional quantum cellular automata must become easier. Moreover our approach may eventually open this (well-advertised in the literature) open problem to an algebraic analysis.

Note also that it is undecidable whether local transition function δ induces a unitary global evolution Δ as soon as the quantum cellular automata is two dimensional. But again this does not have to be the end of it: nothing prevents that there should be a canonical definition of two dimensional quantum cellular automata, easily checked for unitarity, and yet capable of expressing the exact same set of global evolutions. For this purpose algebraic criteria ought to be easier to generalize to higher dimensions.

Acknowledgments

P.J.A would like to thank Shu Yan Chan, Christoph Dürr, Simon Perdrix, and Estrella Sicardi for a number of insightful conversations, and acknowledges the direct help of Thierry Gallay in proving Prop. 6. Many thanks also to Gonzalo Abal, Arturo Lezama, Alejandro Romanelli, Ricardo Siri and above all Philippe Jorrand.

References

1. S. Amoroso, Y. N. Pratt, *Decision procedures for surjectivity and injectivity of parallel maps for tesselations structures*, J. Comp. Syst. Sci., **6**, 448–464, (1972).
2. P. Arrighi, *Algebraic characterizations of unitary linear quantum cellular automata*, arXiv:quant-ph/0512040.
3. T. Boykett, *Efficient exhaustive enumeration of reversible one dimensional cellular automata*, Theoretical Computer Science, (2004).
4. C. Dürr, H. LêThanh, M. Santha, *A decision procedure for well formed quantum cellular automata*, Random Structures and Algorithms, **11**, 381–394, (1997).
5. C. Dürr, M. Santha, *A decision procedure for unitary quantum linear cellular automata*, SIAM J. of Computing, **31**(4), 1076–1089, (2002).
6. R. P. Feynman, *Quantum mechanical computers*, Found. Phys. **16**, 507-531, (1986).
7. P. Hoyer, *Note on linear quantum cellular automata*, manuscript.
8. O. H. Ibarra, T.Jiang, *On the computing power of one-way cellular arrays*, ICALP, 550–562, (1987).
9. R. Landauer, *Irreversibility and heat generation in the computing process*, IBM J. Res. Dev., **5**(183), (1961).
10. D. A. Meyer, *Unitarity in one dimensional nonlinear quantum cellular automata*, arXiv:quant-ph/9604011.
11. M.A. Nielsen, I.L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, (2000).
12. J. Pedersen, *Cellular automata as algebraic systems*, Complex Systems, **6**, 237–250, (1992).
13. B. Schumacher, R. F. Werner, *Reversible quantum cellular automata*, arXiv:quant-ph/0405174.
14. W. Van Dam, *Quantum Cellular Automata*, Master thesis, Department of Mathematics and Computer Science, University of Nijmegen, The Netherlands, (1996).
15. J. Watrous, *On one dimensional quantum cellular automata*, Complex Systems **5**(1), 19–30, (1991).

A Polynomial Time Nilpotence Test for Galois Groups and Related Results

V. Arvind[1] and Piyush P Kurur[2,⋆]

[1] Institute of Mathematical Sciences
C.I.T Campus, Chennai 600 113, India
arvind@imsc.res.in

[2] Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur,
Kanpur 208016, UP, India
ppk@cse.iitk.ac.in

Abstract. We give a deterministic polynomial-time algorithm to check whether the Galois group $\mathrm{Gal}\,(f)$ of an input polynomial $f(X) \in \mathbb{Q}[X]$ is nilpotent: the running time is polynomial in $\mathrm{size}\,(f)$. Also, we generalize the Landau-Miller solvability test to an algorithm that tests if $\mathrm{Gal}\,(f)$ is in Γ_d: this algorithm runs in time polynomial in $\mathrm{size}\,(f)$ and n^d and, moreover, if $\mathrm{Gal}\,(f) \in \Gamma_d$ it computes all the prime factors of $\#\mathrm{Gal}\,(f)$.

1 Introduction

Computing the Galois group of a polynomial is a fundamental problem in algorithmic number theory. Asymptotically, the best known algorithm is due to Landau [3]: on input $f(X)$, it takes time polynomial in $\mathrm{size}\,(f)$ and the order of its Galois group $\mathrm{Gal}\,(f)$. If $f(X)$ has degree n then $\mathrm{Gal}\,(f)$ can have $n!$ elements. Thus, Landau's algorithm takes time exponential in input size. It is a long standing open problem if there is an asymptotically faster algorithm for computing $\mathrm{Gal}\,(f)$. Lenstra's survey [6] discusses this and related problems.

A different kind of problem is to test for a given $f(x)$ if $\mathrm{Gal}\,(f)$ satisfies a specific property without explicitly computing it. Galois's seminal work showing $f(X)$ is solvable by radicals if and only if $\mathrm{Gal}\,(f)$ is solvable is a classic example. Landau and Miller [4] gave a remarkable polynomial-time algorithm for testing solvability of the Galois group without computing the Galois group.

1.1 The Results of This Article

Our main result is a deterministic polynomial-time algorithm for testing if $\mathrm{Gal}\,(f)$ is nilpotent. Although nilpotent groups are a proper subclass of solvable groups, the Landau-Miller solvability test does not give a nilpotence test. Basically, the Landau-Miller test is a method of testing that all composition factors of $\mathrm{Gal}\,(f)$

⋆ Work done when the author was a PhD student at the Institute of Mathematical Sciences, Chennai.

are abelian, which tests solvability. Nilpotence however is a more "global" property, in the sense that it cannot be inferred from properties of the composition factors alone.

We note here that nilpotence testing of Galois groups has been addressed by other researchers with the goal of developing good practical algorithms. For example in [2] an algorithm for nilpotence testing is given which takes worst-case time polynomial in size (f) and $\#\mathrm{Gal}\,(f)$. However, ours is the first algorithm that is provably polynomial time, i.e. runs in time polynomial in size (f), on all inputs. On the other hand, we have not studied the practicality of our algorithm.

Next, we show that the Landau-Miller solvability test can be extended to a polynomial-time algorithm for checking, given $f \in \mathbb{Q}[X]$, if $\mathrm{Gal}\,(f)$ is in Γ_d for constant d. A group G is in Γ_d if there is a composition series $G = G_0 \triangleright \ldots \triangleright G_t = \{1\}$ such that each nonabelian composition factor G_i/G_{i+1} is isomorphic to a subgroup of S_d. The class Γ_d often arises in permutation group algorithms (see e.g. [7]). Moreover, if $\mathrm{Gal}\,(f) \in \Gamma_d$, the prime factors of $\#\mathrm{Gal}\,(f)$ can be found in polynomial time.

1.2 Galois Theory Overview

We quickly recall some Galois theory (see, e.g. [5] for details). Let L and K be fields. If $L \supset K$, we say that L is an extension of K and denote it by L/K. If L/K then L is a vector space over K and by the *degree* of L/K, denoted by $[L : K]$, we mean its dimension. An extension L/K is *finite* if its degree $[L : K]$ is finite. If L/M and M/K are finite extensions then $[L : K] = [L : M].[M : K]$. The polynomial ring $K[X]$ is a unique factorisation domain: every polynomial can be uniquely (up to scalars) written as a product of irreducible polynomials. Let L/K be an extension. An $\alpha \in L$ is *algebraic* over K if $f(\alpha) = 0$ for some $f(X) \in K[X]$. For α algebraic over K, the *minimal polynomial* of α over K is the unique monic polynomial $\mu_\alpha[K](X)$ of least degree in $K[X]$ for which α is a root. We write $\mu_\alpha(X)$ for $\mu_\alpha[K](X)$ when K is understood. Elements $\alpha, \beta \in L$ are *conjugates* over K if they have the same minimal polynomial over K. The smallest subfield of L containing K and α is denoted by $K(\alpha)$.

The *splitting field* K_f of $f \in K[X]$ is the smallest extension of K containing all the roots of f. A finite extension L/K is *normal* if for all irreducible polynomials $f(X) \in K[X]$, either $f(X)$ splits or has no root in L. Any normal extension over K is the splitting field of some polynomial in $K[X]$. An extension L/K is *separable* if for all irreducible polynomials $f(X) \in K[X]$ there are no multiple roots in L. A normal and separable finite extension L/K is a *Galois extension.*

The *Galois group* $\mathrm{Gal}\,(L/K)$ of L/K is the subgroup of automorphisms σ of L that leaves K fixed, i.e. $\sigma(\alpha) = \alpha$ for all $\alpha \in K$. The Galois group $\mathrm{Gal}\,(f)$ of $f \in K[X]$ is $\mathrm{Gal}\,(K_f/K)$. For a subgroup G of automorphisms of L, the *fixed field* L^G is the largest subfield of L fixed by G. We now state the fundamental theorem of Galois.

Theorem 1. [5, Theorem 1.1, Chapter VI] *Let L/K be a Galois extension with Galois group G. There is a one-to-one correspondence between subfields E of L*

containing K and subgroups H of G, given by $E \rightleftharpoons L^H$. The Galois group of $\mathrm{Gal}(L/E)$ *is H and E/K is a Galois extension if and only if H is a normal subgroup of G. If H is a normal subgroup of G and $E = L^H$ then* $\mathrm{Gal}(E/K)$ *is isomorphic to the quotient group G/H.*

1.3 Presenting Algebraic Numbers, Number Fields and Galois Groups

The algorithms we describe take objects like algebraic numbers, number fields etc. as input. We define sizes of these objects. Integers are encoded in binary. A rational r is given by coprime integers a, b such that $r = a/b$. Thus, $\mathrm{size}(r)$ is $\mathrm{size}(a) + \mathrm{size}(b)$. A polynomial $T(X) = a_0 + \ldots + a_n X^n \in \mathbb{Q}[X]$ is given by a list of its coefficients. Thus, $\mathrm{size}(T)$ is defined as $\sum \mathrm{size}(a_i)$.

A *number field* is a finite extension of $\mathbb{Q}$. Let $K/\mathbb{Q}$ be a number field of degree n. By the primitive element theorem [5, Theorem 4.6, Chapter V], there is an algebraic number $\eta \in K$ such that $K = \mathbb{Q}(\eta)$. Such an element is a *primitive element* of $K/\mathbb{Q}$ and its minimal polynomial is a *primitive polynomial.* Let $\mu_\eta(X)$ be the minimal polynomial of η over $\mathbb{Q}$. Then the field K can be written as the quotient $K = \mathbb{Q}[X]/\mu_\eta(X)$. Thus K can be presented by giving a primitive polynomial for $K/\mathbb{Q}$. We can assume that η is an algebraic integer and hence its minimal polynomial $\mu_\eta(X)$ has integer coefficients [5, Proposition 1.1, Chapter VII]. When we say that an algorithm takes a number field K as input we mean that it takes a primitive polynomial $\mu_\eta(X)$ for K as input. Thus the input size for K, which we denote by $\mathrm{size}(K)$, is defined to be $\mathrm{size}(\mu_\eta)$.

Suppose $K = \mathbb{Q}(\eta)$ is presented by $\mu_\eta(X)$. Notice that each $\alpha \in K$ can be expressed as $\alpha = A_\alpha(\eta)$ for a unique polynomial $A_\alpha(X) \in \mathbb{Q}[X]$ of degree less than n. By $\mathrm{size}(\alpha)$ we mean $\mathrm{size}(A_\alpha(X))$. Note that the size of $\alpha \in K$ depends on the primitive element $\eta \in K$. Now, for a polynomial $f(X) = a_0 + \ldots + a_m X^m$ in $K[X]$ we define $\mathrm{size}(f)$ to be $\sum \mathrm{size}(a_i)$.

Let $f(X) \in \mathbb{Q}[X]$ of degree n. For an algorithm purporting to compute $\mathrm{Gal}(f)$, one possibility is that it outputs the complete multiplication table for $\mathrm{Gal}(f)$. However, this could be exponential in $\mathrm{size}(f)$ as $\mathrm{Gal}(f)$ can be as large as $n!$. A succinct presentation of $\mathrm{Gal}(f)$ is as a permutation group acting on the roots of f since elements of $\mathrm{Gal}(f)$ permute the roots of f and are completely determined by their action on the roots of f. Thus, by numbering the roots of f, we can consider $\mathrm{Gal}(f)$ as a subgroup of the symmetric group S_n (note here that $\mathrm{Gal}(f)$ is determined only up to conjugacy as the numbering of the roots is arbitrary). Since any subgroup of S_n has a generator set of size $n-1$ (see e.g. [8]), we can present $\mathrm{Gal}(f)$ in size polynomial in n. Thus, by computing $\mathrm{Gal}(f)$ we mean finding a small generator set for it as a subgroup of S_n. Determining $\mathrm{Gal}(f)$ as a subgroup of S_n is a reasonable way of describing the output. Algorithmically, we can answer several natural questions about a subgroup G of S_n given by generator set in polynomial time. E.g. testing if G is solvable, finding a composition series for G etc. [8].

Previous Complexity Results. As mentioned, the best known algorithm for computing the Galois group of a polynomial is due to Landau [3].

Theorem 2 (Landau). *There is a deterministic algorithm that takes as input a number field K, a polynomial $f(X) \in K[X]$ and a positive integer b in unary, and in time bounded by* $\mathrm{size}(f)$, $\mathrm{size}(K)$ *and b, decides if* $\mathrm{Gal}(K_f/K)$ *has at most b elements, and if so computes* $\mathrm{Gal}(K_f/K)$ *by finding the entire multiplication table of* $\mathrm{Gal}(K_f/K)$ *(and hence also by giving the generating set of* $\mathrm{Gal}(K_f/K)$ *as a permutation group on the roots of $f(X)$).*

The algorithm first computes a primitive element θ of K_f. Determining $\mathrm{Gal}(f)$ amounts to finding the action of the automorphisms on θ. Subsequently, Landau and Miller [4] gave their polynomial-time solvability test.

Theorem 3 (Landau-Miller). *Given $f(X) \in \mathbb{Q}[X]$ there is a deterministic polynomial-time algorithm for testing if* $\mathrm{Gal}(f)$ *is solvable.*

2 Preliminaries

We recall some permutation group theory from Wielandt's book [9]. Let Ω be a finite set. The *symmetric group* $\mathrm{Sym}(\Omega)$ is the group of all permutations on Ω. By a *permutation group on Ω* we mean a subgroup of $\mathrm{Sym}(\Omega)$. For $\alpha \in \Omega$ and $g \in \mathrm{Sym}(\Omega)$, let α^g denote the image of α under the permutation g. For $A \subseteq \mathrm{Sym}(\Omega)$, α^A denotes the set $\{\alpha^g : g \in A\}$. In particular, for $G \leq \mathrm{Sym}(\Omega)$ the *G-orbit* containing α is α^G. The G-orbits form a partition of Ω. Given $G \leq \mathrm{Sym}(\Omega)$ by a generating set A and $\alpha \in \Omega$, there is a polynomial-time algorithm to compute α^G [8].

For $\Delta \subseteq \Omega$ and $g \in \mathrm{Sym}(\Omega)$, Δ^g denotes $\{\alpha^g : \alpha \in \Delta\}$. The setwise stabilizer of Δ, i.e. $\{g \in G : \Delta^g = \Delta\}$, is denoted by G_Δ. If Δ is the singleton set $\{\alpha\}$ we write G_α instead of $G_{\{\alpha\}}$. For any Δ by $G|_\Delta$ we mean G_Δ restricted to Δ. An often used result is the orbit-stabilizer formula stated below [9, Theorem 3.2].

Theorem 4 (Orbit-stabilizer formula). *Let G be a permutation group on* $\mathrm{Sym}(\Omega)$ *and let α be any element of Ω then the order of the group G is given by $\#G = \#G_\alpha . \#\alpha^G$.*

A permutation group G on Ω is *transitive* if there is a single G-orbit. Suppose $G \leq \mathrm{Sym}(\Omega)$ is transitive. Then a non-empty subset Δ of Ω is a *G-block* if for all $g \in G$ either $\Delta^g = \Delta$ or $\Delta^g \cap \Delta = \emptyset$. For every G, Ω is a block and each singleton $\{\alpha\}$ is a block. These are the *trivial blocks* of G. A transitive group G is *primitive* if it has only trivial blocks and it is *imprimitive* if it has nontrivial blocks. A G-block Δ is a *maximal subblock* of a G-block Σ if $\Delta \subset \Sigma$ and there is no G-block Υ such that $\Delta \subset \Upsilon \subset \Omega$. Let Δ and Σ be two G-blocks. A chain $\Delta = \Delta_0 \subset \ldots \subset \Delta_t = \Sigma$ is a *maximal chain* of G-blocks between Δ and Σ if for all i, Δ_i is a maximal subblock of Δ_{i+1}.

For a G-block Δ and $g \in G$, Δ^g is also a G-block such that $\#\Delta = \#\Delta^g$. Let Δ and Σ be two G-blocks such that $\Delta \subseteq \Sigma$. The *Δ-block system of Σ*, is the collection

$$\mathcal{B}(\Sigma/\Delta) = \{\Delta^g : g \in G \text{ and } \Delta^g \subseteq \Sigma\}.$$

The set $\mathcal{B}(\Sigma/\Delta)$ is a partition of Σ. It follows that $\#\Delta$ divides $\#\Sigma$ and by *index* of Δ in Σ, which we denote by $[\Sigma : \Delta]$, we mean $\#\mathcal{B}(\Sigma/\Delta) = \frac{\#\Sigma}{\#\Delta}$. We will use $\mathcal{B}(\Delta)$ to denote $\mathcal{B}(\Omega/\Delta)$. We state the connection between blocks and subgroups [9, Theorem 7.5].

Theorem 5 (Galois correspondence of blocks). *Let $G \leq \mathrm{Sym}(\Omega)$ be transitive and $\alpha \in \Omega$. For $G \geq H \geq G_\alpha$ the orbit $\Delta = \alpha^H$ is a G-block and $G_\Delta = H$. The correspondence $\alpha^H = \Delta \rightleftharpoons G_\Delta = H$ is a one-to-one correspondence between G-blocks Δ containing α and subgroups H of G containing G_α. Furthermore for G-blocks $\Delta \subseteq \Sigma$ we have $[G_\Sigma : G_\Delta] = [\Sigma : \Delta]$.*

Let $G \leq \mathrm{Sym}(\Omega)$ be transitive and Δ and Σ be two G-blocks such that $\Delta \subseteq \Sigma$. Let $G(\Sigma/\Delta)$ denote the group $\{g \in G : \Upsilon^g = \Upsilon \text{ for all } \Upsilon \in \mathcal{B}(\Sigma/\Delta)\}$. We write G^Δ for the group $\mathrm{G}(\Omega/\Delta)$. For any $g \in G_\Sigma$, since g setwise stabilises Σ, g permutes the elements of $\mathcal{B}(\Sigma/\Delta)$. Hence for any $\Upsilon \in \mathcal{B}(\Sigma/\Delta)$ we have $\Upsilon^{g^{-1}\mathrm{G}(\Sigma/\Delta)g} = \Upsilon$. Thus, $\mathrm{G}(\Sigma/\Delta)$ is a normal subgroup of G_Σ. In particular, G^Δ is a normal subgroup of G.

Remark. The following two lemmata are quite standard in permutation group theory. For the reader's convenience we have included short proofs. The following lemma lists important properties of G^Δ.

Lemma 1.

1. *For a G-block $\Delta \subseteq \Sigma$, $\mathrm{G}(\Sigma/\Delta)$ is the largest normal subgroup of G_Σ contained in G_Δ.*
2. *Let Σ be G-block then $G^\Sigma \hookrightarrow \prod_{\Upsilon \in \mathcal{B}(\Sigma)} G|_\Upsilon$.*
3. *Let Δ be a G-subblock of Σ then $\frac{G_\Sigma}{\mathrm{G}(\Sigma/\Delta)}$ is a faithful permutation group on $\mathcal{B}(\Sigma/\Delta)$ and is primitive when Δ is a maximal subblock.*
4. *The quotient group G^Σ/G^Δ can be embedded as a subgroup of $\left(\frac{G_\Sigma}{\mathrm{G}(\Sigma/\Delta)}\right)^l$ for some l.*

Proof. Let $N \subseteq G_\Delta$ be a normal subgroup of G_Σ. Since $\Delta^N = \Delta$, and since G_Σ acts transitively on $\mathcal{B}(\Sigma/\Delta)$, for any $\Upsilon \in \mathcal{B}(\Sigma/\Delta)$ there is a $g \in G_\Sigma$ such that $\Upsilon = \Delta^g$. Therefore, $\Upsilon^N = \Delta^{gN} = \Delta^{Ng} = \Upsilon$ for each $\Upsilon \in \mathcal{B}(\Sigma/\Delta)$. Thus $N \subseteq \mathrm{G}(\Sigma/\Delta)$. Since $\mathrm{G}(\Sigma/\Delta) \trianglelefteq G_\Sigma$ we have proved part 1.

Part 2 directly follows from the definition of G^Σ. Part 3 follows from the fact that $g, h \in G_\Sigma$ have the same action on $\mathcal{B}(\Sigma/\Delta)$ precisely when $g\mathrm{G}(\Sigma/\Delta) = h\mathrm{G}(\Sigma/\Delta)$. The nontrivial $\frac{G_\Sigma}{\mathrm{G}(\Sigma/\Delta)}$-blocks of $\mathcal{B}(\Sigma/\Delta)$ are in 1-1 correspondence with the G-blocks properly between Δ and Σ. Thus, $\frac{G_\Sigma}{\mathrm{G}(\Sigma/\Delta)}$ is primitive if and only if Δ is a maximal subblock of Σ.

For Part 4 notice that we have the group isomorphism

$$\frac{G|_\Upsilon}{\mathrm{G}(\Upsilon/\Delta_\Upsilon)|_\Upsilon} \cong \frac{G_\Upsilon}{\mathrm{G}(\Upsilon/\Delta_\Upsilon)},$$

for each $\Upsilon \in \mathcal{B}(\Sigma)$. As $G^{\Delta} = G^{\Sigma} \cap \prod \mathrm{G}(\Upsilon/\Delta_{\Upsilon})|_{\Upsilon}$ we have

$$G^{\Sigma}/G^{\Delta} \hookrightarrow \prod_{\Upsilon \in \mathcal{B}(\Sigma)} \frac{G|_{\Upsilon}}{\mathrm{G}(\Upsilon/\Delta_{\Upsilon})|_{\Upsilon}} = \prod_{\Upsilon \in \mathcal{B}(\Sigma)} \frac{G_{\Upsilon}}{\mathrm{G}(\Upsilon/\Delta_{\Upsilon})}.$$

Let $g \in G$ such that $\Delta^g = \Delta_{\Upsilon}$. Then, $G_{\Upsilon} = g^{-1}G_{\Sigma}g$ and $\mathrm{G}(\Upsilon/\Delta_{\Upsilon}) = g^{-1}\mathrm{G}(\Sigma/\Delta)g$. Thus, $\frac{G_{\Sigma}}{\mathrm{G}(\Sigma/\Delta)}$ and $\frac{G_{\Upsilon}}{\mathrm{G}(\Upsilon/\Delta_{\Upsilon})}$ are isomorphic, which implies that G^{Σ}/G^{Δ} is isomorphic to a subgroup of $\left(\frac{G_{\Sigma}}{\mathrm{G}(\Sigma/\Delta)}\right)^l$ for some l.

Lemma 2. *Let $G \leq \mathrm{Sym}(\Omega)$ be transitive and $N \trianglelefteq G$. Let $\alpha \in \Omega$. Then the N-orbit α^N is a G-block and the collection of N-orbits is an α^N-block system of Ω under G action. If $N \neq \{1\}$ then $\|\alpha^N\| > 1$. Furthermore, if $G_{\alpha} \leq N \neq G$ then the α^N-block system is nontrivial implying that G is not primitive.*

Proof. Let $\alpha \in \Omega$ and $g \in G$. Then $(\alpha^N)^g = \alpha^{Ng} = \alpha^{gN} = (\alpha^g)^N$. Thus $(\alpha^N)^g$ and α^N are N-orbits, and hence are identical or disjoint. Thus, α^N is a G-block and the N-orbits form a block system. Clearly, if $\alpha^N = \{\alpha\}$ then $N = \{1\}$. Finally, by the Orbit-Stabilizer formula $\#G = \#\Omega \cdot \#G_{\alpha}$ and $\#N = \#\alpha^N \cdot \#G_{\alpha}$. Thus, if $\{1\} \neq N \neq G$ then α^N is a proper G-block.

3 Nilpotence Testing for Galois Groups

First we recall crucial properties of nilpotent transitive permutation groups. These are standard group theoretic facts that we assemble together and, for the sake of completeness, provide proof sketches where necessary. We start with a characterisation of finite nilpotent groups. Let G be a finite group and $p_1, \ldots, p_k$ be the prime factors of $\#G$. For each i, let G_{p_i} be a p_i-Sylow subgroup of G. Then G is *nilpotent* if and only if G is the (internal) direct product $G_{p_1} \times \ldots \times G_{p_k}$. Consequently, G_{p_i} is the unique p_i-Sylow subgroup of G for each i and hence $G_{p_i} \lhd G$.

Lemma 3. *Let $G \leq \mathrm{Sym}(\Omega)$ be transitive and nilpotent, and p be any prime. Then*

(1) The prime p divides $\#G$ if and only if p divides $\#\Omega$.
(2) If $p \mid \#G$ and $\alpha \in \Omega$ then there is a block Σ_p^{α} containing α such that $\#\Sigma_p^{\alpha}$ is the highest power of p that divides $\#\Omega$.
(3) Let Δ be any G-block containing α such that $\#\Delta = p^l$ and suppose p divides $\#G$. Then $\Delta \subseteq \Sigma_p^{\alpha}$. Also, for $q \neq p$, the q-Sylow subgroup of G_{Δ} is given by $G_q \cap G_{\Delta} = G_q \cap G_{\alpha}$.

Proof. Part (1): As G is transitive, $\#\Omega$ divides $\#G$. Hence, each prime factor of $\#\Omega$ divides $\#G$. Let p be a prime factor of $\#G$. For $\alpha \in \Omega$, let $\Sigma_p^{\alpha} = \alpha^{G_p}$. Since G_p is transitive on Σ_p^{α}, it follows from the Orbit-Stabilizer formula that $\#\Sigma_p^{\alpha}$ divides $\#G_p$. Hence $\#\Sigma_p^{\alpha}$ is p^l for some l. Since $G_p \lhd G$, by Lemma 2 it follows that its orbit Σ_p^{α} is G-block which contains more than one element of Ω.

Hence $\#\Sigma_p^\alpha = p^l$ for some $l > 0$. Since p divides the cardinality of a G-block Σ_p^α, p divides $\#\Omega$.

Part (2): From the Galois correspondence of G-blocks (Theorem 5) we have $[\Omega : \Sigma_p^\alpha] = [G : G_{\Sigma_p^\alpha}]$. Notice that p is not a factor of $[G : G_p]$ as G_p is the p-Sylow subgroup of G. Since $G_p \lhd G_{\Sigma_p^\alpha}$ it follows that p is not a factor of $[G : G_{\Sigma_p^\alpha}]$. Hence p is not a factor of $[\Omega : \Sigma_p^\alpha]$.

Part (3): notice that G_Δ is a nilpotent group with the unique normal q-Sylow subgroup $G_q \cap G_\Delta$. Thus, $G_\Delta = \prod_q (G_q \cap G_\Delta)$. By Theorem 5 we have

$$\#\Delta = [G_\Delta : G_\alpha] = \prod_q [G_q \cap G_\Delta : G_q \cap G_\alpha]. \tag{1}$$

Since $G_q \cap G_\Delta$ is a q-group, p divides $[G_q \cap G_\Delta : G_q \cap G_\alpha]$ if and only if $q = p$. However, in Equation 1, $\#\Delta$ is a power of p. This forces $[G_q \cap G_\Delta : G_q \cap G_\alpha] = 1$ for all $q \neq p$. Thus $G_q \cap G_\Delta = G_q \cap G_\alpha$ for $q \neq p$. Therefore, G_Δ is the product group $G_p \cap G_\Delta \times \prod_{q \neq p} G_q \cap G_\alpha$. Since $G_{\Sigma_p^\alpha}$ contains both G_p and G_α we have $G_{\Sigma_p^\alpha} \geq G_\Delta$. Thus, Δ is a G-subblock of Σ_p^α.

We recall a result about permutation p-groups (see e.g. Luks [7, Lemma 1.1]).

Lemma 4. *Let $G \leq \mathrm{Sym}\,(\Omega)$ be a transitive p-group and Δ be a maximal G-block. Then $[\Omega : \Delta] = p$ and $G_\Delta = \mathrm{G}\,(\Omega/\Delta) = G^\Delta$ is a normal group of index p in G.*

The next lemma is an easy consequence of Lemma 4 and it states a useful property of permutation p-groups.

Lemma 5. *Let $H \leq \mathrm{Sym}\,(\Omega)$ be a transitive p-group and $\alpha \in \Omega$. Let $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_t = \Omega$ be any maximal chain of H-blocks. Then*

1. *$[\Delta_{i+1} : \Delta_i] = p$ for all $0 \leq i < t$.*
2. *$H(\Delta_{i+1}/\Delta_i) = H_{\Delta_i}$. Hence, $H_{\Delta_i} \lhd H_{\Delta_{i+1}}$ and the quotient $H_{\Delta_{i+1}}/H_{\Delta_i}$ is cyclic of order p.*

We continue with the notation of Lemma 3. In the next lemma we show that the block structure of transitive nilpotent permutation group G is similar to the block structure of p-groups.

Lemma 6. *Let G be a nilpotent transitive permutation group on Ω and let p be a prime factor of $\#G$. Let Δ be any subset of Σ_p^α. Then Δ is a G-block if and only if Δ is a G_p block (in its transitive action on Σ_p^α).*

Proof. Let H denote the p-Sylow subgroup G_p. Let $\widehat{H}$ denote the product $\prod_{q \neq p} G_q$ of all other Sylow subgroups of G. Then $G = H \times \widehat{H}$. Recall that Σ_p^α is the H-orbit of α.

Firstly, any G-block $\Delta \subseteq \Sigma_p^\alpha$ is an H-block. To prove the converse consider any H-block $\Sigma \subseteq \Sigma_p^\alpha$. Consider the group $G' = H_\Sigma \times (\widehat{H} \cap G_\alpha)$. Notice that the group G' is a subgroup of $G_{\Sigma_p^\alpha}$. Also since G_α is nilpotent, we have $G_\alpha = H_\alpha \times (\widehat{H} \cap G_\alpha)$.

Furthermore since Σ is a H-block, we have $H_\Sigma \geq H_\alpha$. Therefore $G' \geq G_\alpha$ and by the Galois correspondence of blocks (Theorem 5), $\Sigma = \alpha^{G'}$ is a G-block and $G_\Sigma = G'$.

We give a characterisation of nilpotent transitive permutation groups using properties of maximal chains of G-blocks between $\{\alpha\}$ and Σ_p^α which is crucial for our polynomial-time nilpotence test. This characterisation is probably well-known to group theorists. However, as we haven't seen it anywhere, we include a proof.

Theorem 6. *Let $G \leq \mathrm{Sym}(\Omega)$ be a transitive permutation group satisfying properties (1) and (2) of Lemma 3 (which are necessary conditions for nilpotence of G). Fix an $\alpha \in \Omega$. The following statements are equivalent.*

(1) G is nilpotent.
(2) For each prime factor p of $\#G$, every maximal chain of G-blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_m = \Sigma_p^\alpha$ has the property that $[\Delta_{i+1} : \Delta_i] = p$, G_{Δ_i} is a normal subgroup of $G_{\Delta_{i+1}}$, and p does not divide the order of G/G^{Δ_m}.
(3) For each prime p dividing $\#G$, there is a maximal chain of G-blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_m = \Sigma_p^\alpha$ with the property that $[\Delta_{i+1} : \Delta_i] = p$, G_{Δ_i} is a normal subgroup of $G_{\Delta_{i+1}}$, and p does not divide the order of G/G^{Δ_m}.

Proof. Clearly (2) implies (3). It suffices to show that (3) implies (1) and (1) implies (2).

To see that (3) implies (1) it is enough to show that each Sylow subgroup of G is normal. To this end, let p be a prime factor of $\#G$ and let $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_m = \Sigma_p^\alpha$ be a maximal chain of G-blocks having the properties mentioned in (3).

Firstly, since $\mathrm{G}(\Delta_{i+1}/\Delta_i)$ is the largest normal subgroup of $G_{\Delta_{i+1}}$ that is contained in G_{Δ_i} (part 1 of Lemma 1), (3) implies that $G_{\Delta_i} = \mathrm{G}(\Delta_{i+1}/\Delta_i)$. Furthermore it follows from Lemma 1 that there is a positive integer l_i for each i such that the quotient group $G^{\Delta_{i+1}}/G^{\Delta_i}$ is embeddable in an l_i-fold product of copies of $\frac{G_{\Delta_{i+1}}}{\mathrm{G}(\Delta_{i+1}/\Delta_i)} = G_{\Delta_{i+1}}/G_{\Delta_i}$. Since $[G_{\Delta_{i+1}} : G_{\Delta_i}] = p$ it follows that $G^{\Delta_{i+1}}/G^{\Delta_i}$ is a p-group for each i. As $\#G^{\Delta_m} = \prod_{i=0}^{m-1}[G^{\Delta_{i+1}} : G^{\Delta_i}]$, G^{Δ_m} is also a p-group. Since $G^{\Delta_m} \lhd G$ and p does not divide $[G : G^{\Delta_m}]$ it follows that G^{Δ_m} is a normal p-Sylow subgroup of G. The nilpotence of G follows as this holds for all prime factors of $\#G$.

Next, we show that (1) implies (2). Suppose G is nilpotent. Let p be a prime factor of $\#G$ and $\alpha \in \Omega$. Let H be the p-Sylow subgroup G_p of G and let $\widehat{H} = \prod_{q \neq p} G_q$ be the product of all its other Sylow subgroups. Let $\{\alpha\} = \Delta_0 \subset \Delta_1 \subset \ldots \subset \Delta_m = \Sigma_p^\alpha$ be any maximal chain of G-blocks between α and Σ_p^α. It follows from Lemma 6 that the chain $\{\Delta\}_{0 \leq i \leq m}$ is a maximal chain of G_p-blocks. By Lemma 5 we have $[\Delta_{i+1} : \Delta_i] = p$, $H_{\Delta_i} \lhd H_{\Delta_{i+1}}$, and $H_{\Delta_{i+1}}/H_{\Delta_i}$ is cyclic of order p. The group $G_{\Delta_i} = H_{\Delta_i} \times \widehat{H}_{\Delta_i}$ and $G_{\Delta_{i+1}} = H_{\Delta_{i+1}} \times \widehat{H}_{\Delta_{i+1}}$. Also since $\widehat{H}_{\Delta_i}$ is the product of q-Sylow subgroups of H_{Δ_i} where q varies over all prime factors of $\#G$ different from p, it follows from Lemma 3 that $\widehat{H}_{\Delta_i} = \widehat{H}_\alpha$. Therefore $G_{\Delta_i} \lhd G_{\Delta_{i+1}}$ and quotient group $G_{\Delta_{i+1}}/G_{\Delta_i} \cong H_{\Delta_{i+1}}/H_{\Delta_i}$. The group G/G^{Δ_m} acts faithfully on $\mathcal{B}(\Omega/\Delta_m)$ and is transitive under this action. Since $p \nmid [\Omega : \Delta_m]$, p cannot divide the order of G/G^{Δ_m} (Lemma 3).

The following lemma is important for the nilpotence testing algorithm. If G is nilpotent then, for each prime factor p of $\#G$, the lemma implies that no matter how the maximal chain of blocks Δ_i of Theorem 6 is constructed, it must terminate in Σ_p^α.

Lemma 7. *Let G be a transitive nilpotent permutation group on Ω. Let p be any prime dividing $\#G$. Let Δ be any G-block such that $\#\Delta = p^l$ for some integer $l \geq 0$. Let m be the highest power of p that divides $\#\Omega$. If $l < m$ then we have*

1. *There exists a G-block Σ such that Δ is a maximal G-subblock of Σ and $[\Sigma : \Delta] = p$.*
2. *For all G-blocks Σ such that Δ is a maximal G-subblock of Σ and $[\Sigma : \Delta] = p$, G_Δ is a normal subgroup of G_Σ.*

Proof. Since $\#\Delta$ is p^l it follows that Δ is a G-subblock of Σ_p^α (Lemma 3). It follows from Lemma 6 that Δ is a G_p-block on the transitive action of G_p on Σ_p^α. Furthermore if $l < m$ there is a G_p-block Σ (and hence by Lemma 6 a G-block) such that $\Sigma_p^\alpha \supseteq \Sigma \supset \Delta$ and $[\Sigma : \Delta] = p$. This proves part 1.

Let $\alpha \in \Delta$. It follows from Lemma 3 that for $q \neq p$ the q-Sylow subgroup of G_Σ and G_Δ are both $G_q \cap G_\alpha$. Let $\widehat{G}_p$ be $\prod_{q \neq p} G_q$. The groups G_Σ and G_Δ are $(G_p \cap G_\Sigma) \times (\widehat{G}_p \cap G_\alpha)$ and $(G_p \cap G_\Delta) \times (\widehat{G}_p \cap G_\alpha)$ respectively. Moreover, $G_p \cap G_\Sigma$ and $G_p \cap G_\Delta$ are p-groups with index $[G_p \cap G_\Sigma : G_p \cap G_\Delta] = [G_\Sigma : G_\Delta] = [\Sigma : \Delta] = p$. Therefore, $G_p \cap G_\Delta$ is normal in $G_p \cap G_\Sigma$. Thus, $G_\Delta = (G_p \cap G_\Delta) \times (\widehat{G}_p \cap G_\alpha)$ is normal in $G_\Sigma = (G_p \cap G_\Sigma) \times (\widehat{G}_p \cap G_\alpha)$ and $\frac{G_\Sigma}{G_\Delta} = \frac{G_p \cap G_\Sigma}{G_p \cap G_\Delta}$ is isomorphic to $\mathbb{Z}_p$.

3.1 The Nilpotence Test

Given $f(X) \in \mathbb{Q}[X]$ our goal is to test if Gal (f) is nilpotent. We can assume that $f(X)$ is irreducible. For, otherwise we can compute the irreducible factors of $f(X)$ over $\mathbb{Q}$ using the LLL algorithm, and perform the nilpotence test on each distinct irreducible factor. This suffices because nilpotent groups are closed under products and subgroups. Let G be Gal (f). We consider G as a subgroup of Sym (Ω), where Ω is the set of roots of $f(X)$. Since f is irreducible, G is transitive on Ω.

For any G-block Δ, let $\mathbb{Q}_\Delta$ be the fixed field of the splitting field $\mathbb{Q}_f$ under the automorphisms of G_Δ. Let Δ be a G-block containing α. Since $G_\Delta \geq G_\alpha$, $\mathbb{Q}_\Delta$ is a subfield of $\mathbb{Q}_{\{\alpha\}} = \mathbb{Q}(\alpha)$.

We describe the main idea. By Theorem 6, G is nilpotent if and only if for all primes p that divide the order of G, there is a maximal chain of G-blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_m$ satisfying conditions of part (3) of Theorem 6. We show these conditions can be verified in polynomial time once the tower of fields $\mathbb{Q}(\alpha) = \mathbb{Q}_{\Delta_0} \supset \ldots \supset \mathbb{Q}_{\Delta_m}$ are known. Thus, for testing nilpotence of G we will first need a polynomial-time algorithm that computes $\mathbb{Q}_{\Delta_i}$. The following theorem is essentially due to Landau and Miller [4] restated in a form suitable for our application.

Theorem 7. *Let $f(X) \in \mathbb{Q}[X]$ be irreducible, $G = \mathrm{Gal}(f)$ be its Galois group and Ω be the set of roots of f. Let $\Delta \subseteq \Omega$ be any G-block and $\alpha \in \Delta$. There is an algorithm that given a primitive polynomial $\mu_\Delta(X) \in \mathbb{Q}[X]$ of $\mathbb{Q}_\Delta$, runs in time polynomial in* size(f) *and* size(μ_Δ) *and computes a primitive polynomial $\mu_\Sigma(X) \in \mathbb{Q}[X]$ of $\mathbb{Q}_\Sigma$ for all G-blocks Σ such that Δ is a maximal block of Σ. Moreover* size(μ_Σ) *is at most a polynomial in* size(f) *and is independent of* size(μ_Δ).

We now give a high level description of the nilpotence testing algorithm.

```
  Input: A polynomial $f(X) \in \mathbb{Q}[X]$ of degree $n$
  Output: "Accept" if Gal$(f)$ is nilpotent;"Reject" otherwise
  Using the Landau-Miller test verify that Gal$(f)$ is solvable;
1 Compute the set $P$ of all the prime factors of #Gal$(f)$;
  Let $G \leq \mathrm{Sym}(\Omega)$ denote the Galois group of $f$, where $\Omega$ is the set of roots of $f$.
2 for every $p \in P$ do
      if $p$ does not divide $n$ then
          print Reject
      end
      Let $m$ be the highest power of $p$ dividing $n$.
3     Attempt to compute the tower $\mathbb{Q}_{\Delta_m} \subset \ldots \subset \mathbb{Q}_{\Delta_0}$ for a maximal chain of $G$-
      blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_m$ such that $[\mathbb{Q}_{\Delta_{i+1}} : \mathbb{Q}_{\Delta_i}] = p$.
4     if Step 3 fails or $\mathbb{Q}_{\Delta_{i+1}}$ is not normal over $\mathbb{Q}_{\Delta_i}$ then
          print Reject
      end
      Let $\mu_{\Delta_m}(X)$ be the primitive polynomial for $\mathbb{Q}_{\Delta_m}$
5     if $p$ divides #Gal$(\mu_{\Delta_m})$ then
          print Reject
      end
  end
  print Accept
```

Algorithm 1. Nilpotence test

We prove that Algorithm 1 runs in polynomial time. For steps 1 and 5 note that for polynomials f with solvable Galois groups, as a byproduct of the Landau-Miller test [4], the prime factors of #Gal(f) can be found in polynomial time (see also Theorem 11). We now explain how step 3 can be done in polynomial time using Theorem 7. We construct $\mathbb{Q}_{\Delta_i}$ inductively starting with $\mathbb{Q}_{\Delta_0} = \mathbb{Q}(\alpha)$. Assume we have computed $\mathbb{Q}_{\Delta_i}$. Using Theorem 7 we compute $\mathbb{Q}_\Sigma$ for each G-block Σ containing Δ_i as a maximal G-subblock. Among them choose a $\mathbb{Q}_\Sigma$ for which $[\Sigma : \Delta_i] = p$ and let $\mathbb{Q}_{\Delta_{i+1}}$ be $\mathbb{Q}_\Sigma$. The inductive construction of $\mathbb{Q}_{\Delta_{i+1}}$ from $\mathbb{Q}_{\Delta_i}$ can be done in time bounded by a polynomial in size(f). Putting it together we have the following proposition.

Proposition 1. *Algorithm 1 runs in time polynomial in* size(f).

We now argue its correctness. Part (1) of Theorem 6 implies that if G is nilpotent then Algorithm 1 accepts. Conversely, suppose the algorithm accepts. Then for each prime p dividing $\#G$ we have a maximal chain of G-blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_m$ such that $\mathbb{Q}_{\Delta_i}/\mathbb{Q}_{\Delta_{i+1}}$ are normal extensions for each $0 \le i < m$ (this we verify in step 4 of Algorithm 1). Recall that $\mathbb{Q}_{\Delta_i}$ is the fixed field of $\mathbb{Q}_f$ w.r.t. G_{Δ_i}. Hence by checking $\mathbb{Q}_{\Delta_i}/\mathbb{Q}_{\Delta_{i+1}}$ is a normal extension we have verified that $G_{\Delta_i} \lhd G_{\Delta_{i+1}}$. Also, the splitting field of the primitive polynomial $\mu_{\Delta_m}(X)$ is the normal closure of $\mathbb{Q}_{\Delta_m}$ over $\mathbb{Q}$. It follows from Lemma 1 and Theorem 1 that $\mathrm{Gal}\,(\mu_{\Delta_m})$ is G^{Δ_m}. Hence, by checking p does not divide $\#\mathrm{Gal}\,(\mu_\Delta)$ we have verified that p does not divide $\#G/G^{\Delta_m}$. Thus, we have verified that the maximal chain of G-blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_m$ satisfies the conditions of Part(3) of Theorem 6 implying that G is nilpotent. Putting it all together we have the following theorem.

Theorem 8. *There is a polynomial-time algorithm that takes $f \in \mathbb{Q}[X]$ as input and tests if* $\mathrm{Gal}\,(f)$ *is nilpotent.*

4 Generalising the Landau-Miller Solvability Test

In this section we show that the Landau-Miller solvability test can be adapted to test if the Galois group of $f(X) \in \mathbb{Q}[X]$ is in Γ_d for constant d. Note that for $d < 5$, Γ_d is the class of solvable groups and hence our result is a generalisation of the result of Landau-Miller [4]. We first recall a well-known bound on the size of primitive permutation groups in Γ_d.

Theorem 9 ([1]). *Let $G \le S_n$ be a primitive permutation group in Γ_d for a constant d. Then $\#G \le n^{O(d)}$.*

Theorem 10. *For constant $d > 0$, there is an algorithm that takes as input $f(X) \in \mathbb{Q}[X]$ and in time polynomial in* $\mathrm{size}\,(f)$ *and $n^{O(d)}$ decides whether* $\mathrm{Gal}\,(f)$ *is in Γ_d.*

Proof. We sketch the proof. Assume without loss of generality that $f(X)$ is irreducible. Let $G = \mathrm{Gal}\,(f)$ as a subgroup of $\mathrm{Sym}\,(\Omega)$, where Ω is the set of roots of f. Let $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_t = \Omega$ be any maximal chain of G-blocks. The series $\{1\} = G^{\Delta_0} \lhd \ldots \lhd G^{\Delta_t} = G$ gives a normal series for G. By closure properties of Γ_d, $G \in \Gamma_d$ iff $\frac{G^{\Delta_{i+1}}}{G^{\Delta_i}} \in \Gamma_d$ for each i. If G is in Γ_d so are $G_{\Delta_{i+1}}$ and $\mathrm{G}\,(\Delta_{i+1}/\Delta_i)$ and hence their quotient $\frac{G_{\Delta_{i+1}}}{\mathrm{G}(\Delta_{i+1}/\Delta_i)}$. On the other hand since $\frac{G^{\Delta_{i+1}}}{G^{\Delta_i}}$ is isomorphic to a subgroup of $\left(\frac{G_{\Delta_{i+1}}}{\mathrm{G}(\Delta_{i+1}/\Delta_i)}\right)^l$ for some l (Lemma 1), $\frac{G^{\Delta_{i+1}}}{G^{\Delta_i}} \in \Gamma_d$ if $\frac{G_{\Delta_{i+1}}}{\mathrm{G}(\Delta_{i+1}/\Delta_i)} \in \Gamma_d$. Hence $G \in \Gamma_d$ iff $\frac{G_{\Delta_{i+1}}}{\mathrm{G}(\Delta_{i+1}/\Delta_i)}$ is in Γ_d for each i . We give a polynomial-time algorithm to verify the above fact for some maximal chain of G-blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_t = \Omega$.

First, by Theorem 7we compute $K_i = \mathbb{Q}_{\Delta_i}$ for a maximal chain of G-blocks $\{\alpha\} = \Delta_0 \subset \ldots \subset \Delta_t = \Omega$. Let L_i be the fixed field of $\mathbb{Q}_f$ with respect to the automorphisms of $\mathrm{G}\,(\Delta_{i+1}/\Delta_i)$ then L_{i+1} is the normal closure of K_i over K_{i+1}. This follows because $\mathrm{G}\,(\Delta_{i+1}/\Delta_i)$ is the largest proper normal subgroup of

$G_{\Delta_{i+1}} = \mathrm{Gal}\left(\mathbb{Q}_f/\mathbb{Q}_{\Delta_{i+1}}\right)$. Hence $\mathrm{Gal}\,(L_{i+1}/K_{i+1})$ is $\frac{G_{\Delta_{i+1}}}{\mathrm{G}(\Delta_{i+1}/\Delta_i)}$, and it suffices to check that each $\mathrm{Gal}\,(L_i/K_i)$ is in Γ_d.

The group $\frac{G_{\Delta_{i+1}}}{\mathrm{G}(\Delta_{i+1}/\Delta_i)}$ acts faithfully and primitively on $\Omega' = \mathcal{B}\,(\Delta_{i+1}/\Delta_i)$, by Lemma 1 and since Δ_i is a maximal subblock of Δ_{i+1}. If $G \in \Gamma_d$ then $[L_{i+1} : K_{i+1}] = \#\mathrm{Gal}\,(L_{i+1}/K_{i+1}) \leq n^{O(d)}$ and degrees $[L_i : \mathbb{Q}]$ are all less than $n^{O(d)}$. We can use Theorem 2 to compute $\mathrm{Gal}\,(L_i/K_i)$ as a multiplication table in time polynomial in $\mathrm{size}\,(f)$ and n^d for each i. We then verify that $\mathrm{Gal}\,(L_i/K_i) \in \Gamma_d$ by computing a composition series for it and checking that each composition factor is in Γ_d. At any stage in the computation of $\mathrm{Gal}\,(L_i/K_i)$ if the sizes of the fields becomes too large, i.e. larger than the bound of Theorem 9 we abort the computation and decide that $\mathrm{Gal}\,(f)$ is not in Γ_d. Clearly, these steps can be done in polynomial time.

It follows from the proof of Theorem 10 that a prime p divides $\#\mathrm{Gal}\,(f)$ if and only if it divides $[L_i : K_i]$ for some $1 \leq i \leq t$.

Theorem 11. *Given $f(X) \in \mathbb{Q}[X]$ with Galois group in Γ_d there is an algorithm running in time polynomial in* $\mathrm{size}\,(f)$ *and n^d that computes all the prime factors of* $\#\mathrm{Gal}\,(f)$.

References

1. L. Babai, P. J. Cameron, and P. P. Pálfy. On the order of primitive groups with restricted nonabelian composition factors. *Journal of Algebra*, 79:161–168, 1982.
2. P. Fernandez-Ferreiros and M. A. Gomez-Molleda. Deciding the nilpotency of the galois group by computing elements in the centre. *Mathematics of Computation*, 73(248), 2003.
3. S. Landau. Polynomial time algorithms for galois groups. In J. Fitch, editor, *EUROSAM 84 Proceedings of International Symposium on Symbolic and Algebraic Computation*, volume 174 of *Lecture Notes in Computer Sciences*, pages 225–236. Springer, July 1984.
4. S. Landau and G. L. Miller. Solvability by radicals is in polynomial time. *Journal of Computer and System Sciences*, 30:179–208, 1985.
5. S. Lang. *Algebra*. Addison-Wesley Publishing Company, Inc, third edition, 1999.
6. H. W. Lenstra Jr. Algorithms in algebraic number theory. *Bulletin of the American Mathematical Society*, 26(2):211–244, April 1992.
7. E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.
8. E. M. Luks. Permutation groups and polynomial time computations. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 11:139–175, 1993.
9. H. Wielandt. *Finite Permutation Groups*. Academic Press, New York, 1964.

The Multiparty Communication Complexity of Exact-T: Improved Bounds and New Problems

Richard Beigel[1], William Gasarch[2,⋆], and James Glenn[3]

[1] Temple University, Dept. of Computer and Information Sciences, 1805 N Broad St Fl 3, Philadelphia, PA 19122
professorB@gmail.com

[2] University of Maryland, Dept. of Computer Science and Institute for Advanced Computer Studies, College Park, MD 20742
gasarch@cs.umd.edu

[3] Dept. of Computer Science, Loyola College in Maryland, 4501 N. Charles St, Baltimore, MD 21210
jglenn@cs.loyola.edu

Abstract. Let $x_1, \ldots, x_k$ be n-bit numbers and $T \in \mathbb{N}$. Assume that $P_1, \ldots, P_k$ are players such that P_i knows all of the numbers *except* x_i. They want to determine if $\sum_{j=1}^{k} x_j = T$ by broadcasting as few bits as possible. In [7] an upper bound of $O(\sqrt{n})$ bits was obtained for the $k = 3$ case, and a lower bound of $\omega(1)$ for $k \geq 3$ when $T = \Theta(2^n)$. We obtain (1) for $k \geq 3$ an upper bound of $k + O((n + \log k)^{1/(\lfloor \lg(2k-2) \rfloor)})$, (2) for $k = 3$, $T = \Theta(2^n)$, a lower bound of $\Omega(\log \log n)$, (3) a generalization of the protocol to abelian groups, (4) lower bounds on the multiparty communication complexity of some regular languages, and (5) empirical results for $k = 3$.

1 Introduction

Multiparty communication complexity was first defined in [7] and was used to obtain lower bounds on branching programs (BPs). It has been used to get additional lower bounds and tradeoffs for BPs [1, 5], lower bounds on data structures [5], time-space tradeoffs for restricted TMs [1], and unconditional pseudorandom generators for logspace [1].

Def 1.1 Let $f : \{\{0,1\}^n\}^k \rightarrow \{0,1\}$. Assume, for $1 \leq i \leq k$, P_i has all of the inputs *except* x_i. Let $d(f)$ be the total number of bits broadcast in the optimal deterministic protocol for f. This is called the *multiparty communication complexity* of f. This scenario is called *the forehead model.*

The multiparty communication complexity of the following function was used [7] to obtain superlinear lower bounds on constant width BPs (later improved by [2, 4, 14]).

⋆ Partially supported by NSF grant CCR-01-05413.

Def 1.2 Let $k, n, T \in \mathbb{N}$. (T stands for Target.) Let $f_{k,T} : \{\{0,1\}^n\}^k \to \{0,1\}$ be defined as

$$f_{k,T}(x_1, \ldots, x_k) = \begin{cases} 1 & \text{if } \sum_{j=1}^{k} x_j = T; \\ 0 & \text{otherwise.} \end{cases}$$

We refer to $f_{k,T}$ as *the Exact-T problem.*

Determining $d(f_{k,T})$ is equivalent to a problem in combinatorics. From this one obtains:

1. $d(f_{3,T}) = O(\sqrt{n})$.
2. For all k, for $T = \Theta(2^n)$, $d(f_{k,T})$ is not constant in n.

This paper contains the following.

1. New upper and lower bounds on $d(f_{k,T})$:
   (a) For $k \geq 4$, $d(f_{k,T}) \leq k + O((n + \log k)^{1/(\lfloor \lg(2k-2) \rfloor)})$.
   (b) For $k = 3$, for $T = \Theta(2^n)$, $d(f_{3,T}) \geq \Omega(\log \log n)$. The proof uses a Ramsey-theoretic lemma (Lemma 5.1).
2. A group-theoretic version of the Exact-T problem that we denote $f^{\mathcal{G}}_{k,T}$.
   (a) Bounds on $d(f^{\mathcal{G}}_3)$ yield bounds on $d(f_{k,T})$.
   (b) For all finite abelian groups $\mathcal{G}$ of size g, $d(f^{\mathcal{G}}_{3,T}) \geq \Omega(\log \log \log g)$.
   (c) For almost all finite abelian groups $\mathcal{G}$, a nontrivial protocol for $f^{\mathcal{G}}_{k,T}$.
   (d) $d(f^{\mathbb{Z}_m}_{k,T}) \leq k + O((\log m + \log k)^{1/(\lfloor \lg(2k-2) \rfloor)})$.
3. Application: We use the lower bound on $d(f^{\mathcal{G}}_k)$ to obtain lower bounds on the multiparty communication complexity of several regular languages.
4. Empirical results: We have some empirical results about 3-free sets that lead to concrete upper bounds on $d(f_{3,T})$ for $T = 2^n$.

Notation 1.3 If $T \in \mathbb{N}$ then $[T]$ denotes the set $\{1, \ldots, T\}$.

Def 1.4 $f \leq^{O(1)}_{\text{cc}} g$ if there exists a protocol for f that has the following properties. (1) The protocol may invoke a protocol for g once on an input of length $O(n)$, (2) before and after the invocation, the players may broadcast $O(1)$ bits. $f \equiv^{O(1)}_{\text{cc}} g$ if $f \leq^{O(1)}_{\text{cc}} g$ and $g \leq^{O(1)}_{\text{cc}} f$. Note that $\leq^{O(1)}_{\text{cc}}$ is transitive and that if $f \leq^{O(1)}_{\text{cc}} g$ then $d(f) \leq d(g) + O(1)$.

2 Multiparty Communication Complexity and Combinatorics

In this section we review the connections between the multiparty communication complexity of $f_{3,T}$ and combinatorics that was first established in [7]. We also review the upper and lower bounds that they obtained.

Def 2.1 Let $c, k, T \in \mathbb{N}$ with $k \geq 3$.

1. A *proper c-coloring of* $[T]^{k-1}$ is a function $C : [T]^{k-1} \rightarrow [c]$ such that there do not exist $x_1, \ldots, x_{k-1} \in [T]$ and $\lambda \in \mathbb{Z} - \{0\}$ with (1) for all i, $x_i + \lambda \in [T]$, and (2) $C(x_1, x_2, x_3, \ldots, x_{k-1}) = C(x_1 + \lambda, x_2, x_3, \ldots, x_{k-1}) = \cdots = C(x_1, x_2, x_3, \ldots, x_{k-1} + \lambda)$
2. Let $\chi_k(T)$ be the least c such that there is a proper c-coloring of $[T]^{k-1}$.

Theorem 2.2 *[7]*

1. $d(f_{k,T}) \leq k - 1 + \lceil \lg(\chi_k(T) + 1) \rceil = k + \lg(\chi_k(T)) + O(1)$.
2. *If* $x_1, \ldots, x_k \in \{0, \ldots, T\}$ *then* $d(f_{k,T}) \geq \lg(\chi_k(\lfloor \frac{T}{k} \rfloor)) + \Omega(1)$.

Def 2.3

1. A k-AP is an arithmetic progression of length k.
2. A set $A \subseteq [T]$ is k*-free* if there do not exist any k-AP's in A.
3. Let $\mathrm{r}_k(T)$ be the size of the largest k-free subset of $[T]$.

The next theorem states combinatorial facts that are needed for the upper and lower bounds, and then the bounds themselves.

Theorem 2.4 *[7]*

1. $d(f_{k,T}) \leq k - 1 + \lg(\chi_k(T)) \leq k - 1 + \left\lceil \lg(\frac{2kT \ln(kT)}{\mathrm{r}_k(kT)}) + 1 \right\rceil = k + O\big(\log(\frac{kT \log(kT)}{\mathrm{r}_k(kT)})\big)$
2. *For all* k, $\chi_k(2^n)$ *is an increasing function of* n.
3. *If* $T = \Theta(2^n)$ *then* $d(f_{k,T}) = \omega(1)$.

Chandra, Furst, and Lipton used the fact that there are 3-free sets of $[T]$ of size $T 2^{-O(\log T)^{1/2}}$ (due to [6], but see [13] for a constructive version and [8] for an exposition) to obtain the following.

Corollary 2.5 $d(f_{3,T}) \leq O(\sqrt{\log T})$. *When* $T = \Theta(2^n)$, $d(f_{3,T}) = O(\sqrt{\log T}) = O(\sqrt{n})$.

3 New Upper Bounds

The following lemma yields large k-free sets. We will use these sets to obtain new explicit upper bounds for $\chi_k(T)$ when $k \geq 4$, which will in turn yield new explicit upper bounds on $d(f_{k,T})$. This lemma was first proven in [15] but see also [12].

Lemma 3.1 $\mathrm{r}_k(T) \geq T 2^{-O((\log T)^{1/(\lfloor \lg(2k-2) \rfloor)})}$.

Theorem 3.2

1. $d(f_{k,T}) \leq k + O((\log kT)^{1/(\lfloor \lg(2k-2) \rfloor)})$.
2. *If* $T = \Theta(2^n)$ *then* $d(f_{k,T}) = k + O((n + \log k)^{1/(\lfloor \lg(2k-2) \rfloor)})$.

Proof: (1) Follows from Theorem 2.4 and Lemma 3.1. (2) Follows from part 1 of this theorem. ∎

4 Group Theoretic Version

We define a group-theoretic version of the Exact-T problem.

Def 4.1 Let $\mathcal{G} = (G, \odot)$ be a group.

1. Let $f^{\mathcal{G}}_{k,T} : G^k \to \{0,1\}$ be defined by

$$f^{\mathcal{G}}_k(x_1, \ldots, x_k) = \begin{cases} 1 & \text{if } \bigodot_{j=1}^{k} x_j = ID; \\ 0 & \text{otherwise.} \end{cases}$$

2. A *$\mathcal{G}$-proper c-coloring of* G^{k-1} is a function $C : G^{k-1} \to [c]$ such that there do not exist $x_1, \ldots, x_{k-1} \in G$ and $\lambda \in G - \{ID\}$ with $C(x_1, \ldots, x_{k-1}) =$

$$C(x_1 \odot \lambda, x_2, x_3, \ldots, x_{k-1}) = \cdots = C(x_1, x_2, x_3, \ldots, x_{k-1} \odot \lambda).$$

3. $\chi^*_k(\mathcal{G})$ be the least c such that there is a $\mathcal{G}$-proper c-coloring of G^{k-1}.

The proof of the following theorem is a modification of a proof from [7], hence we omit it.

Theorem 4.2 *If $\mathcal{G}$ is a finite abelian group then* $\lg(\chi^*_k(\mathcal{G})) + \Omega(1) \le d(f^{\mathcal{G}}_k) \le k + \lg(\chi^*_k(\mathcal{G})) + O(1)$.

Note 4.3 In Theorem 4.2 $d(f^{\mathcal{G}}_k) \ge \lg(\chi^*_k(\mathcal{G})) + \Omega(1)$. Chandra, Furst, and Lipton obtained $d(f_{k,T}) \ge \lg(\chi_k(\lfloor \frac{T}{k} \rfloor)) + \Omega(1)$. They have a factor of $\frac{1}{k}$ and we do not because in the group case, for any $x_1, \ldots, x_{k-1} \in G$ there is an $x \in G$ such that $f^{\mathcal{G}}_k(x_1, \ldots, x_{k-1}, x) = 1$; by contrast, there are $x_1, \ldots, x_{k-1} \in [T]$ such that, for all $x \in [T]$, $f_{k,T}(x_1, \ldots, x_{k-1}, x) = 0$.

The next lemma shows a relation between $d(f^{\mathcal{G}}_k)$ and $d(f_{k,T})$ that we will use to obtain bounds on one from bounds on the other.

Def 4.4 $\mathbb{Z}_T$ is the group with set $\{0, 1, \ldots, T-1\}$ under modular addition.

Lemma 4.5 *Let $T \in \mathbb{N}$ and $k \ge 3$. Then the following hold.*

1. $d(f_{k,T}) \le d(f^{\mathbb{Z}_T}_k) + O(1)$.
2. $d(f^{\mathbb{Z}_T}_k) \le d(f_{k,T}) + O(\log k)$.

5 Lower Bounds

5.1 An $\Omega(\log \log \log g)$ Lower Bound for $d(f^{\mathbb{Z}_T}_3)$ and $d(f_{3,T})$

The following combinatorial lemma will allow us to prove a lower bound on $d(f^{\mathcal{G}}_3)$ for a variety of $\mathcal{G}$. This lemma is a reworking of a theorem of Graham and Solymosi [10].

Lemma 5.1 *There exist absolute constants g_0, d_0 such that the following is true. Let $\mathcal{G} = (G, \odot)$ be any finite abelian group and let $g = |G|$. If $g \geq g_0$ and $c \leq d_0 \log\log g$ then there are no $\mathcal{G}$-proper c-colorings of $G \times G$. Hence $\chi_3^*(\mathcal{G}) \geq \Omega(\log\log g)$.*

Proof: Assume that C is a $\mathcal{G}$-proper c-coloring of $G \times G$. We will find sets $X_1, Y_1 \subseteq G$ such that C restricted to $X_1 \times Y_1$ uses $c-1$ colors. We will iterate this process to obtain X_c, Y_c such that C restricted to $X_c \times Y_c$ uses 0 colors. Hence $|X_c| = 0$ which will yield $c = \Omega(\log\log g)$.

Let $X_0 = G$, $Y_0 = G$, $h_0 = |X_0| = |Y_0| = g$, $COL_0 = [c]$. At stage s the subset will be $X_s \times Y_s$, the size of X_s will be $h_s = |X_s| = |Y_s|$, and COL_s will be the colors used by $X_s \times Y_s$.

Assume X_s, Y_s, h_s are defined and inductively $COL_s = [c-s]$ (we will be renumbering to achieve this). Partition $X_s \times Y_s$ into sets P_a indexed by $a \in G$ defined by $P_a = \{(x,y) \in X_s \times Y_s \mid x \odot y = a\}$. (Think of P_a as the ath anti-diagonal.) There exists an a such that $|P_a| \geq \lceil h_s^2/g \rceil$. There exists a color, which we will take to be $c-s$ by renumbering, such that at least $\lceil \lceil h_s^2/g \rceil /c \rceil$ of the elements of P_a are colored $c-s$. (We could use $c-s$ in the denominator but we do not need to.) Let $m = \lceil \lceil h_s^2/g \rceil /c \rceil$. Let $\{(x_1,y_1), \ldots, (x_m,y_m)\}$ be m elements of P_a such that, for $1 \leq i \leq m$, $C(x_i,y_i) = c-s$.

Claim 1: For all $i \neq j$, $x_i \neq x_j$ and $y_i \neq y_j$.

Proof: If $x_i = x_j$ then $x_j \odot y_j = a = x_i \odot y_i = x_j \odot y_i$. Hence $y_j = y_i$. Therefore $(x_i,y_i) = (x_j,y_j)$. This contradicts P_a having m distinct points. The proof that $y_i \neq y_j$ is similar. *End of Proof of Claim 1*

Claim 2: For all $i \neq j$, $C(x_i,y_j) \neq c-s$.

Proof: If $C(x_i,y_j) = c-s$ then $C(x_i,y_j) = C(x_i,y_i) = C(x_j,y_j) = c-s$. If $\lambda = (a^{-1} \odot x_j \odot y_i)$ then $C(x_i,y_j) = C(x_i \odot \lambda, y_j) = C(x_i, y_j \odot \lambda)$. This violates C being a proper coloring. *End of Proof of Claim 2*

Let

$$\begin{aligned} h_{s+1} &= m' = \lceil m/3 \rceil \\ X_{s+1} &= \{x_1, \ldots, x_{m'}\} \\ Y_{s+1} &= \{y_{m+1-m'}, \ldots, y_m\} \\ COL_{s+1} &= [c-(s+1)] \end{aligned}$$

Note that, by Claim 2 above, $\{C(x,y) \mid x \in X_{s+1}, y \in Y_{s+1}\} \subseteq COL_{s+1}$. We iterate the process c times to obtain X_c, Y_c such that COL restricted to $X_s \times Y_s$ uses 0 colors.

We have $h_0 = g$ and

$$h_{s+1} = \left\lceil \left\lceil \left\lceil \frac{h_s^2}{g} \right\rceil /c \right\rceil /3 \right\rceil \geq \frac{h_s^2}{3cg}.$$

One can easily show that $h_s \geq \frac{g}{(3c)^{2^s-1}}$.

Taking $s = c$ we obtain $h_c \geq \frac{g}{(3c)^{2^c-1}}$. Hence there is a set of h_c^2 points that are 0-colored. Therefore $h_c < 1$. This yields $c = \Omega(\log\log g)$. ∎

Theorem 5.2 *If $\mathcal{G}$ is a finite abelian group then $d(f_3^{\mathcal{G}}) \geq \Omega(\log\log\log|G|)$.*

Proof: By Lemma 5.1 $\chi_3^*(\mathcal{G}) \geq \Omega(\log\log|G|)$. By Theorem 4.2, $d(f_3^{\mathcal{G}}) \geq \lg(\chi_3^*(\mathcal{G})) \geq \Omega(\log\log\log|G|)$. ∎

From Theorem 5.2 and Lemma 4.5 we obtain the following.

Theorem 5.3 *Let $T \in \mathbb{N}$. $d(f_{3,T}) \geq \Omega(\log\log\log T) - O(\frac{(\log k)2^n}{T})$. If $T = \Theta(2^n)$ then $d(f_{3,T}) \geq \Omega(\log\log n) - O(\log k)$.*

5.2 An $\omega(1)$ Lower Bound for General $\mathcal{G}$ and k

From Lemma 4.5 and Theorem 2.4 we obtain the following.

Theorem 5.4 $d(f_k^{\mathbb{Z}_m}) = \omega(1)$.

For other groups we cannot use Lemma 4.5 and hence we develop other techniques.

Def 5.5 Fix k. The phrase $d(f_k^{\mathcal{G}}) = \omega(1)$ means that, for all constants d, there exists g_0, such that for all finite abelian groups G of size $g \geq g_0$, $d(f_k^{\mathcal{G}}) \geq d$.

Def 5.6 $\mathrm{PART}_{n,k} : \{\{0,1\}^n\}^k \to \{0,1\}$ is the following function. Interpret the input as k subsets of $\{1,\ldots,n\}$. Output 1 if these sets form a partition of $\{1,\ldots,n\}$, and 0 otherwise.

Tesson [17, 18] proved the following. He used the Hales-Jewitt Theorem (see [9]) which is why the bound is $\omega(1)$ instead of something more concrete. We use this lemma to obtain $d(f_k^{\mathcal{G}}) = \omega(1)$.

Lemma 5.7 *For all k, $d(\mathrm{PART}_{n,k}) \geq \omega(1)$.*

Lemma 5.8 *Let $k \geq 3$. Let $h_1,\ldots,h_m \geq 2$. Let $\mathcal{G} = \mathbb{Z}_{h_1} \times \cdots \times \mathbb{Z}_{h_m}$ For all k, $d(\mathrm{PART}_{m,k}) \leq d(f_k^{\mathcal{G}}) + O(1)$.*

Proof sketch: One can show that $\mathrm{PART}_{n,k} \leq_{\mathrm{cc}}^{O(1)} f_k^{\mathcal{G}}$. (Recall Definition 1.4.) ∎

Lemma 5.9 *If $\mathcal{G}_1$ and $\mathcal{G}_2$ are groups and $k \geq 3$ then $d(f_k^{\mathcal{G}_1}) \leq d(f_k^{\mathcal{G}_1 \times \mathcal{G}_2})$.*

Theorem 5.10 *For all d, k there exists g_0 such that for all finite abelian groups $\mathcal{G}$ with $|G| \geq g_0$, $d(f_k^{\mathcal{G}}) \geq d$. In short, the bigger the group, the larger $d(f_k^{\mathcal{G}})$, without bound.*

6 Application to Multiparty Communication Complexity of Regular Languages

In this section we use Theorems 5.2 and Theorem 5.10 to obtain lower bounds on the multiparty communication complexity of many regular languages.

The 2-party communication complexity of regular languages has been defined and solved completely [16, 20, 19]. The multiparty communication complexity of regular languages (defined initially in [16]) still has many open problems. The standard problem in this field is as follows.

Def 6.1 Let L be a regular language and k be the number of players. $R_{k,L}$ is the following problem.

1. Let $x = a_1 a_2 \cdots a_{kn}$ be a string such that $(\forall i)[a_i \in \Sigma \cup \{\epsilon\}]$.
2. Player P_i gets all a_j such that $j \not\equiv i \pmod{k}$.
3. The players want to determine if $a_1 a_2 \cdots a_{kn} \in L$.

Notation 6.2 The multiparty communication complexity of $R_{k,L}$ is denoted $d(R_{k,L})$.

Notation 6.3 Let $\sigma \in \Sigma$, $m \in \mathbb{N}$, and $r \in \mathbb{N}$ such that $0 \le r \le m-1$.

1. $\#_\sigma(w)$ is the number of σ in w.
2. $L_{\sigma,r,m} = \{w \mid \#_\sigma(w) \equiv r \pmod{m}\}$.

Lemma 6.4 *Let $k, r, m \in \mathbb{N}$ be such that $0 \le r \le m-1$. Let $|\Sigma| \ge 2$, $\sigma \in \Sigma$, and $L = L_{\sigma,r,m}$. Then $f_k^{\mathbb{Z}_m} \le_{\rm cc}^{O(1)} R_{k,L}$. (Recall Definition 1.4.)*

Proof: We show $f_{k,r}^{\mathbb{Z}_m} \le_{\rm cc}^{O(1)} R_{k,L}$. It is easy to show that $f_k^{\mathbb{Z}_m} \equiv_{\rm cc}^{O(1)} f_{k,r}^{\mathcal{G}}$, hence we will have $f_k^{\mathbb{Z}_m} \le_{\rm cc}^{O(1)} R_{k,L}$.

We map $(q_1, \ldots, q_k)$ to a string w of length km such that $f_{k,r}^{\mathbb{Z}_m}(q_1, \ldots, q_k) = 1$ iff $\#_\sigma(w) \equiv r \pmod{m}$.

For each i, $1 \le i \le k$, there are m positions that are $\equiv i \pmod{k}$. Map to a string such that q_i of those positions are σ and the rest are not σ.

If w is the resulting word then $\#_\sigma(w) = \sum_{i=1}^{k} q_i$. Hence $q_1 + \cdots + q_k \equiv r \pmod{m}$ iff $w \in L$. ∎

Theorem 6.5 *Let $k, r, m \in \mathbb{N}$ such that $0 \le r \le m-1$. Let $|\Sigma| \ge 2$ and $\sigma \in \Sigma$. Let $L = L_{\sigma,r,m}$.*

1. *$d(R_{3,L}) \ge \Omega(\log\log\log m)$.*
2. *For all $k \ge 4$, $\omega(1) \le d(R_{k,L})$.*

Proof: By Lemma 6.4 $d(f_k^{\mathcal{G}}) \le d(R_{k,L})$.
1) By Theorem 5.2 $d(f_3^{\mathcal{G}}) = \omega(\log\log\log m)$. Hence $d(R_{3,L}) = \omega(\log\log\log m)$.
2) By Theorem 5.10 $d(f_3^{\mathcal{G}}) = \omega(1)$. Hence $d(R_{k,L}) = \omega(1)$. ∎

7 Upper Bounds

7.1 Upper Bounds for $\mathcal{G} = \mathbb{Z}_m$

The proofs in this section are a reworking of those in [7].

Notation 7.1 If $\mathcal{G} = (G, \odot)$ is a group and $d \in G$, $k \in \mathbb{N}$, then d^k means $d \odot \cdots \odot d$ where there are k d's.

Def 7.2 Let $\mathcal{G} = (G, \odot)$ be a group. Let $T = |G|$.

1. A k-AP$^{\mathcal{G}}$ is a multiset of the form $\{a, a \odot d, a \odot d^2, \ldots, a \odot d^{k-1}\}$ where $a, d \in G$.
2. A set $A \subseteq G$ is *k-free* if there do not exist any k-AP$^{\mathcal{G}}$'s in A.
3. Let $\mathrm{r}_k(\mathcal{G})$ be the size of the largest k-free subset of G.

Lemma 7.3 *If $\mathcal{G}$ is a finite abelian group then $\chi_k^*(\mathcal{G}) \le O(\frac{|G|\log(|G|)}{\mathrm{r}_k(\mathcal{G})})$. $\chi_k^*(\mathcal{M}) \le O(\frac{T\log(T)}{\mathrm{r}_k(\mathcal{M})})$.*

Lemma 7.4 *Let $T \in \mathbb{N}$. $\chi_k^*(\mathbb{Z}_T) \le 2^{O((\log T)^{1/(\lfloor \lg(2k-2) \rfloor)})}$.*

Theorem 7.5 *Let $T \in \mathbb{N}$. $d(f_k^{\mathbb{Z}_T}) \le k + O((\log kT)^{1/(\lfloor \lg(2k-2) \rfloor)})$.*

7.2 Upper Bounds for General Groups

If $\mathcal{G}$ has low characteristic then it does not have large k-free sets, so the technique of Lemma 7.3 does not improve upon a trivial upper bound. Hence we use other techniques.

Lemma 7.6 *Let $\mathcal{G}_1 = (G_1, \odot_1)$ and $\mathcal{G}_2 = (G_2, \odot_2)$ be any two finite groups. Let n_1, n_2 be such that, for $i = 1, 2$, $2^{n_i - 1} < |G_i| \le 2^{n_i}$. Assume $n_1 \le n_2$. We represent elements of G_i by a subset of $\{0,1\}^{n_2}$. Let $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$.*

1. $\chi_3^*(\mathcal{G}) \le 2^{n_2} = \Theta(|G_2|)$.
2. $d(f_3^{\mathcal{G}}) \le 2 + n_2 = \Theta(\log(|G_2|))$.

Proof:
1) Let $\oplus : \{0,1\}^{n_2} \times \{0,1\}^{n_2} \to \{0,1\}^{n_2}$ be the bitwise XOR function. For $i = 1, 2$ Let ID_i be the identify in $\mathcal{G}_i$.

We show that $\chi_3^*(\mathcal{G}) \le 2^{n_2}$. Let $(a_1, a_2), (b_1, b_2) \in G_1 \times G_2$. Let $C((a_1, a_2), (b_1, b_2)) = a_1 \oplus b_2 \in \{0,1\}^{n_2}$. It is easy to show that C is a $\mathcal{G}_1 \times \mathcal{G}_2$-proper coloring.
2) Since $\chi_3^*(\mathcal{G}) \le 2^{n_2}$ we have, from Theorem 4.2, $d(f_3^{\mathcal{G}}) \le 2+n_2$. ∎

Lemma 7.7 *If $\mathcal{G} = \mathcal{G}_1 \times \cdots \times \mathcal{G}_a$ then $\chi_k^*(\mathcal{G}) \le \prod_{i=1}^{a} \chi_k^*(\mathcal{G}_i)$.*

Proof: Let C_i be a proper $\chi_k^*(\mathcal{G}_i)$-coloring of $\mathcal{G}_i^{k-1}$. Let C be the coloring

$$C((z_1^1,\ldots,z_a^1),\ldots,(z_1^{k-1},\ldots,z_a^{k-1})) = C_1(z_1^1,\ldots,z_1^{k-1})\cdots C_a(z_a^1,\ldots,z_a^{k-1}).$$

It is routine to check that this is a $\mathcal{G}$-proper coloring. ∎

Lemma 7.8 *If $\mathcal{G}$ is a finite abelian group, $k \ge 3$, then $d(f_k^{\mathcal{G}}) \le d(f_{k-1}^{\mathcal{G}})$.*

Theorem 7.9 *For all $k \ge 3$ there exists $\alpha < 1$ such that for all finite abelian groups $\mathcal{G}$, $d(f_k^{\mathcal{G}}) < k + \alpha \lg(|G|) + O(1)$. (There is a nontrivial protocol for $f_k^{\mathcal{G}}$.)*

Proof sketch: Fix k. Let $\mathcal{G}$ be a finite abelian group of size g. By the classification of finite abelian groups, $\mathcal{G} = \mathbb{Z}_{h_1} \times \cdots \times \mathbb{Z}_{h_b}$ for some factorization $g = \prod_{i=1}^{b} h_i$. We assume $h_1 \le \cdots \le h_b$. By Lemma 7.4, for all i, $\chi_k^*(\mathbb{Z}_{h_i}) \le 2^{O((\lg h_i)^{1/(k-1)})}$.

There are two cases. They depend on a constant β to be picked later.
Case 1: $b \le \beta \lg g$. By Lemma 7.7, $\chi_k^*(\mathcal{G}) = \chi_k^*(\mathbb{Z}_{h_1})\cdots\chi_k^*(\mathbb{Z}_{h_b}) \le \prod_{i=1}^{b} 2^{O((\lg h_i)^{1/(k-1)})}$.

So $\lg(\chi_k^*(\mathcal{G})) \le \sum_{i=1}^{b} O((\lg h_i)^{1/(k-1)}) \le O(\sum_{i=1}^{b}((\lg h_i)^{1/(k-1)})$.

The quantity $\sum_{i=1}^{b}(\lg h_i)^{1/(k-1)}$, where $\prod_{i=1}^{b} h_i = g$, is maximized when $h_1 = \cdots = h_b = g^{1/b}$. Hence $\sum_{i=1}^{b}(\lg h_i)^{1/(k-1)} \le b^{(k-2)/(k-1)}(\lg g)^{1/(k-1)}$. Pick $\beta < 1$ such that $\alpha = c\beta^{(k-2)/(k-1)} < 0.9$.
Case 2: $b \ge \beta \lg g$. Since all $h_i \ge 2$ we have $\prod_{i=1}^{b/2} h_i > 2^{b/2} \ge 2^{\beta \lg g/2} = g^{\beta/2}$. So $\prod_{i=b/2+1}^{b} h_i < g^{1-(\beta/2)}$. Let $\mathcal{G}_1 = \mathbb{Z}_{h_1} \times \cdots \times \mathbb{Z}_{h_{b/2}}$ and $\mathcal{G}_2 = \mathbb{Z}_{h_{b/2+1}} \times \cdots \times \mathbb{Z}_{h_b}$ Note that $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2$ and that $|G_2| \ge |G_1|$. By Lemmas 7.8 and 7.6, $d(f_k^{\mathcal{G}}) \le d(f_3^{\mathcal{G}})\lg(|G_2|) + O(1) \le \lg(g^{1-\beta/2}) + O(1) \le (1-\beta/2)\lg g + O(1)$. Since $0 < \beta < 1$ we have $(1-(\beta/2)) < 1$. Take $\alpha = \max\{0.9, 1-(\beta/2)\}$. ∎

8 Open Problems

1. If $T = \Theta(2^n)$ then $\Omega(\log\log n) \le d(f_{3,T}) \le \sqrt{n}$. Improve either side.
2. If $T = \Theta(2^n)$ and $k \ge 4$ then $\omega(1) \le d(f_{k,T}) \le k + O((n+\log k)^{1/(\lfloor \lg(2k-2) \rfloor)})$. Improve either side.
3. Theorem 4.2 shows $\lg(\chi_k^*(\mathcal{G})) + \Omega(1) \le d(f_k^{\mathcal{G}}) \le k + \lg(\chi_k^*(\mathcal{G})) + \Omega(1)$. For a variety of abelian groups $\mathcal{G}$ estimate $\chi_k^*(\mathcal{G})$.
4. What happens to $d(f_k^{\mathcal{G}})$ if G is nonabelian? A Monoid? Infinite?
5. Empirical studies could be done to see if there are colorings that use substantially fewer than the number of colors induced by 3-free sets.

Acknowledgments

We would like to thank Jozsef Solymosi, Paul Pudlak for refs; and Adam Bender, Walid Gomma, Dov Gordon, Jon Katz, Clyde Kruskal, Martin Ma, Matthew Mah, Brian Postow, and Arkady Yerukhimovich for proofreading.

References

[1] Babai, Nisan, and Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *JCSS*, 45, 1992.

[2] L. Babai, P. Pudlak, V. Rodl, and E. Szemeredi. Lower bounds to the complexity of symmetric Boolean functions. *TCS*, 74:313–323, 1990.

[3] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *JCSS*, 38, 1989.

[4] D. Barrington and H. Straubing. Superlinear lower bounds for bounded width branching programs. *JCSS*, 50, 1995.

[5] P. Beame and E. Vee. Time-space tradeoffs, multiparty communication complexity and nearest neighbor problems. In *34th STOC*, 2002.

[6] F. Behrend. On set of integers which contain no three in arithmetic progression. *Proc. of the Nat. Acad. of Sci. (USA)*, 23:331–332, 1946.

[7] A. Chandra, M. Furst, and R. Lipton. Multiparty protocols. In *15th STOC*, pages 94–99, 1983.

[8] W. Gasarch and J. Glenn. Finding large sets without arithmetic progressions of length three: An empirical view, 2005.

[9] R. Graham, A. Rothchild, and J. Spencer. *Ramsey Theory*. Wiley, 1990.

[10] R. Graham and J. Solymosi. Monochromatic equilateral right triangles on the integer grid, 2005. See Solymosi's website

[11] E. Kushilevitz and N. Nisan. *Comm. Comp.* Camb Univ. Press, 1997.

[12] I. Laba and M. T. Lacey. On sets of integers not containing long arithmetic. progressions, 2001. See arxiv.org

[13] L. Moser. On non-averaging sets of integers. *Canadian Journal of Mathematics*, 5:245–252, 1953.

[14] P. Pudlak. A lower bound on complexity of branching programs. In *MFCS84*, pages 480–489, 1984.

[15] R. Rankin. Sets of integers containing not more than a given number of terms in an arithmetic. progressions. *Proc. of the Royal Soc. of Edinburgh Sect. A 65*, 332–344, 1960–1961.

[16] J.-F. Raymond, P. Tesson, and D. Therien. An algebraic approach to communication complexity. In *25th ICALP*, vol. 1443 of LNCS pages 29–40. 1998. Also at Tesson Website.

[17] P. Tesson. *Computational complexity questions related to finite monoids and semigroups.* PhD thesis, McGill University, 2003.

[18] P. Tesson. An application of the Hales-Jewitt Theorem to multiparty communication complexity, 2004. See Gasarch's Ramsey Website.

[19] P. Tesson and D. Therien. Monoids and computations. *Int. J. of Algebra and Computation*, pages 115–163, 2004. www.cs.mcgill.ca/ptesso.

[20] P. Tesson and D. Therien. Complete classification of the communication complexity of regular languages. *TOCS*, pages 135–159, 2005.

[21] I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Application.* SIAM, 2000.

Appendix: Empirical Results

Gasarch and Glenn [8] produced tables of sizes of 3-free sets. The table below was produced using their software. The table gives n, a lower bound on $\mathrm{r}_3(3N)$,

$n = \lg N$, and $d(f_{3,T}) = 3 + \left\lceil \lg\left(\frac{6N \ln(3N)}{\mathrm{r}_3(3N)} + 1\right) \right\rceil$ (from Theorem 2.4.1). We also give the ratio of $d(f_{3,T})$ to $\sqrt{n}$ since $O(\sqrt{n})$ is what the analysis gives. We only show an excerpt of the table – the full table will be in the journal version.

1. The lowest value where we know that the main protocol beats the trivial one is around 10^4. This is fairly small.
2. The ratio seems to be around 0.31. This is fairly small.

N	$\mathrm{r}_3(3N)$	df	n	$\lceil\sqrt{n}\rceil$	ratio
10	10	7	4	2	0.286
100	48	9	7	3	0.333
1000	210	10	10	4	0.4
10000	1024	12	14	4	0.333
100000	4096	13	17	5	0.385
10^6	16384	15	20	5	0.333
10^7	65536	16	24	5	0.312
10^8	262144	18	27	6	0.333
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
10^{60}	4.54×10^{49}	47	200	15	0.319
10^{61}	3.61×10^{50}	47	203	15	0.319
10^{62}	2.87×10^{51}	47	206	15	0.319
10^{63}	2.28×10^{52}	48	210	15	0.312
10^{64}	1.81×10^{53}	48	213	15	0.312
10^{65}	1.44×10^{54}	48	216	15	0.312

Crochemore Factorization of Sturmian and Other Infinite Words

Jean Berstel[1] and Alessandra Savelli[1,2]

[1] Institut Gaspard Monge (IGM)
Université de Marne-la-Vallée
[2] Dipartimento di Elettronica e Informazione
Politecnico di Milano
{berstel, savelli}@univ-mlv.fr

Abstract. The Crochemore factorization was introduced by Crochemore for the design of a linear time algorithm to detect squares in a word. We give here the explicit description of the Crochemore factorization for some classes of infinite words, namely characteristic Sturmian words, (generalized) Thue-Morse words, and the period doubling sequence.

1 Introduction

In a seminal paper, Ziv and Lempel [7] defined several factorizations of finite words related to information theory and text processing. Several years later, Crochemore ([2,4,3]) introduced a similar factorization of words as a key tool in the design of a linear algorithm checking words for square freeness.

We study here both Ziv-Lempel and Crochemore factorization of special classes of infinite words, such as Sturmian words and some automatic words. It appears that these factorizations can be expressed by a closed formula in many significant examples. The proof of these formulas require some insight in the combinatorial structure of the infinite words considered.

Some factorizations are quite surprising. As an example, the Ziv-Lempel factorization of the Fibonacci word will be shown to be

$$f = a|b|aa|bab|aabaa|\cdots$$

This is precisely the so called singular factorization introduced by Wen and Wen ([8], see also [1]) in a completely different context.

Ziv-Lempel and Crochemore factorizations have similar properties. Both can be computed in linear time by preprocessing the suffix tree of the word. Furthermore, the number of factors in both factorizations are closely related: the number of factors of the Crochemore factorization is at most twice the number of factors of the Ziv-Lempel factorization. However, there are examples of factorizations which differ significantly infinitely many times.

In this paper we study the behavior of the Crochemore factorization in the case of some of the most known classes of words, i.e., characteristic Sturmian words, the Thue-Morse word, and the period doubling sequence.

As we shall see, the Crochemore factorization (or c-factorization for short) of special infinite words can be described explicitly, and it reflects the structure of these words.

The c-factorization $c(x)$ of a word x is defined as follows. Each factor of $c(x)$ is either a fresh letter, or it is a maximal factor of x already occurring in the prefix of the word; more formally, the *c-factorization* $c(x)$ of a word x is

$$c(x) = (x_1, x_2, \ldots, x_m, x_{m+1}, \ldots)$$

where x_m is the longest prefix of $x_m x_{m+1} \cdots$ occurring twice in $x_1 x_2 \cdots x_m$, or x_m is a letter a if a does not occur in $x_1 \cdots x_{m-1}$. For example, the c-factorization of $x = ababaab$ is (a, b, aba, ab), since aba occurs twice in $ababa$.

Note that the the c-factorization of a word differs slightly from the well known Ziv-Lempel factorization [7] (or z-factorization), so that these two factorizations are in general not comparable. The *z-factorization* $z(x)$ of a word x is

$$z(x) = (y_1, y_2, \ldots, y_m, y_{m+1}, \ldots)$$

where y_m is the shortest prefix of $y_m y_{m+1} \cdots$ which occurs only once in the word $y_1 y_2 \cdots y_m$.

For example, let x be the word $x = aabaaccbaabaabaa$. The c-factorization and the z-factorization of x are:

$$c(x) = (a, a, b, aa, c, c, baa, baabaa)$$
$$z(x) = (a, ab, aac, cb, aabaab, aa).$$

We shall discuss the relation between these factorizations in more detail in the final section.

The c-factorization has an interesting behavior in all of the well known infinite words we have considered. For example, take the Fibonacci word

$$\mathbf{f} = abaababaabaab \cdots$$

defined inductively by $f_{-1} = b$, $f_0 = a$, and $f_{n+2} = f_{n+1} f_n$. The c-factorization of f is

$$c(\mathbf{f}) = (a, b, a, aba, baaba, \ldots) = (a, b, a, \widetilde{f_2}, \widetilde{f_3}, \ldots)$$

Observe that each of the factors (except the first three) is the reverse the finite Fibonacci word f_n. We will see that a similar result holds for characteristic Sturmian words (Theorem 1 below).

The c-factorization is closely related to two other factorizations of the Fibonacci word. The first is the factorization into factors which are exactly the prefixes f_n, that is

$$h(\mathbf{f}) = (a, b, a, ab, aba, abaab, \ldots) = (a, f_{-1}, f_0, f_1, f_2, \ldots)$$

The other is the Wen and Wen factorization (also called the singular factorization), in which the ith factor has the same length as f_{i-1}, thus resulting

$$w(\mathbf{f}) = (a, b, aa, bab, aabaa, \ldots) = (a, w_0, w_1, w_2, w_3, \ldots)$$

Note that in the Wen and Wen factorization the ith factor is a palindrome, and is the only factor of f of length $|f_{i-1}|$ that is not a conjugate of f_{i-1}.

The three factorizations can be visualized through the following scheme:

$h:$	a	b	a	$a\ b$	$a\ b\ a$	$a\ b\ a\ a\ b$	$\cdots$
$w:$	a	b	$a\ a$	$b\ a\ b$	$a\ a\ b\ a\ a$	$\cdots$	
$c:$	a	b	a	$a\ b\ a$	$b\ a\ a\ b\ a$	$\cdots$	

The relation between these factorizations is the following. Factors of h and w satisfy

$$bf_{2i} = w_{2i}a \text{ and } af_{2i+1} = w_{2i+1}b,$$

while factors of w and c satisfy

$$aw_{2i} = \widetilde{f}_{2i}b \text{ and } bw_{2i+1} = \widetilde{f}_{2i+1}a.$$

The c-factorization on Fibonacci word is a particular case of a more general result we obtained for the c-factorization on standard Sturmian words.

Theorem 1. *Let* $\mathbf{s}$ *be the standard Sturmian word defined as the limit of*

$$s_{-1} = b, s_0 = a, \text{ and } s_n = s_{n-1}^{d_n} s_{n-2},$$

where $d_i > 0$ *for each* i*. Then*

$$c(\mathbf{s}) = (a, a^{d_1-1}, b, a^{d_1}\widetilde{s}_1^{\,d_2-1}, \widetilde{s}_2^{\,d_3}, \widetilde{s}_3^{\,d_4}, \ldots, \widetilde{s}_n^{\,d_{n+1}}, \ldots)$$

Similar results hold for other familiar infinite words, such as the Thue-Morse and the period doubling sequence. However we do not yet have a full characterization of the c-factorization of automatic words.

The paper is organized as follows. Section 2 contains definitions and statements of theorems. Section 3 sketches the proof for Sturmian words, Section 4 sketches the proof for a family of Thue-Morse sequences and for the period doubling sequence, and Section 5 makes some comparison of Crochemore and Ziv-Lempel factorizations.

2 Basic Definitions and Main Results

Let A be an alphabet and A^* the the set of finite words on A. For any finite word $x = a_1a_2\cdots a_n$, $|x|$ denotes the length n of x and $\widetilde{x}$ denotes the reverse word $a_n \cdots a_2a_1$ of x. If A is a two-letter alphabet $A = \{a, b\}$, then $\overline{x}$ is the image of x of the morphism defined by $\overline{a} = b$ and $\overline{b} = a$. If $x = y\alpha$, with $x, y \in A^*$ and $\alpha \in A$, we denote by x' the word $x' = y\overline{\alpha}$.

A *factorization* of a finite word x is a sequence $(x_1, x_2, \ldots, x_n)$ such that $x = x_1x_2\cdots x_n$. Analogously, a factorization of an infinite word x is a sequence $(x_1, x_2, \ldots)$ such that $x = x_1x_2\cdots$. A recent introduction to factorizations of words can be found in [6]. The c-factorization $c(x)$ of a word x can be constructively defined by induction on the length of x as follows. If x is a letter, his c-factorization is $c(x) = (x)$. Otherwise, let $x = y\alpha$ with $\alpha \in A$, and $c(y) = (u_1, \ldots, u_k)$. The c-factorization $c(x)$ of x is then

$$c(x) = \begin{cases} (u_1, \ldots, u_k, a) & \text{if } u_k a \text{ is not a factor of } y \\ (u_1, \ldots, u_k a) & \text{otherwise .} \end{cases}$$

The Ziv-Lempel factorization of a word x is

$$z(x) = (y_1, y_2, \ldots, y_m, y_{m+1}, \ldots)$$

where y_m is the shortest prefix of $y_m y_{m+1} \cdots$ which occurs only once in the word $y_1 y_2 \cdots y_m$.

Let $(d_1, d_2, \ldots)$ be an infinite sequence of integers (called a *directive sequence* such that $d_1 \geq 0$ and $d_i > 0$ for $i > 1$, and let $\{s_n\}_{n \geq 0}$ be the infinite sequence of words defined by

$$s_{-1} = b,\ s_0 = a, \text{ and } s_n = s_{n-1}^{d_n} s_{n-2}.$$

It is easy to see that this sequence converges to the infinite word s that is called a *standard Sturmian word.*

Note that the Sturmian word defined by a directive sequence $(0, d_2, d_3, \ldots)$ is obtained from the Sturmian word defined by $(d_2, d_3, \ldots)$ by changing each letter a with a letter b and viceversa, so that in the rest of this paper we will only refer to directive sequences with $d_1 > 0$.

With a result that is very similar to that obtained by de Luca in [5], we have that any standard Sturmian word has a particular decomposition in reverse finite words s_n:

$$\mathbf{s} = \widetilde{s_0}^{\,d_1} \widetilde{s_1}^{\,d_2} \cdots$$

The c-factorization of standard Sturmian words stated in Theorem 1

$$c(\mathbf{s}) = (a, a^{d_1 - 1}, b, a^{d_1} \widetilde{s_1}^{\,d_2 - 1}, \widetilde{s_2}^{\,d_3}, \widetilde{s_3}^{\,d_4}, \ldots, \widetilde{s_n}^{\,d_{n+1}}, \ldots)$$

is then clearly closely related to that decomposition.

Let τ be the Thue-Morse morphism on a two-letter alphabet defined by

$$\tau(a) = ab,\ \tau(b) = ba,$$

and let $\{t_n\}_{n \geq 0}$ be the infinite sequence of words such that $t_0 = a$ and $t_n = \tau(t_{n-1})$. This sequence converges to the well known *Thue-Morse infinite word*

$$\mathbf{t} = abbabaabbaababba \cdots$$

Each factor in the c-factorization of $\mathbf{t}$ can be obtained from the previous ones by applying the morphism τ as stated in the following Theorem.

Theorem 2. *The c-factorization $c(\mathbf{t}) = (c_1, c_2, \ldots)$ of the Thue-Morse sequence is $(a, b, b, ab, a, abba, c_7, c_8, \ldots)$ where $c_{n+2} = \tau(c_n)$ for every $n \geq 7$.*

The Thue-Morse word can be generalized in several different ways. We consider here the infinite word $\mathbf{t^{(m)}}$ on a m-letter alphabet $A = \{a_1, a_2, \ldots, a_m\}$ obtained as the limit of the morphism τ_m defined by

$$\tau_m(a_i) = a_i a_{i+1} \cdots a_m a_1 \cdots a_{i-1} \quad (i = 1, \ldots, m).$$

A result that is very similar and even better than that obtained for the Thue-Morse word is the following.

Theorem 3. *Let* $c(\mathbf{t^{(m)}}) = (c_1^{(m)}, c_2^{(m)}, \ldots)$ *be the c-factorization of the generalized Thue-Morse word* $\mathbf{t^{(m)}}$ *with* $m \geq 3$*. Then* $c_{n+2(m-1)}^{(m)} = \tau_m(c_n)$ *for every* $n > m$*.*

It shows that the c-factorization of the infinite words $\mathbf{t^{(m)}}$ for $m > 2$ are even more regular than in the binary case.

Finally, let δ be the morphism on a two-letter alphabet defined by

$$\delta(a) = ab, \quad \delta(b) = aa,$$

and let $\{q_n\}_{n\geq 0}$ be the infinite sequence of words such that

$$q_0 = a \text{ and } q_{n+1} = \delta(q_n).$$

The limit $\mathbf{q}$ of this sequence is the *period doubling sequence.*
We will denote the reverse of q_i by ${q_i}^R$ and the reverse of q_i' by ${q_i}^S$ (we recall that q_n' is q_n with just the last letter changed to its opposite).

Similarly to the case of standard Sturmian words, the period doubling sequence is the composition of the reverse of the finite period doubling sequence words q_n:

$$\mathbf{q} = {q_0}^R {q_1}^R {q_2}^R \cdots.$$

The c-factorization of $\mathbf{q}$ reflects indeed this property, as stated in the following Theorem, observing that the equality $q_{n+1}^R = q_n^S q_n^R$ holds for each n.

Theorem 4. *Let* $\mathbf{q}$ *be the doubling period sequence. The c-factorization of* $\mathbf{q}$ *is*

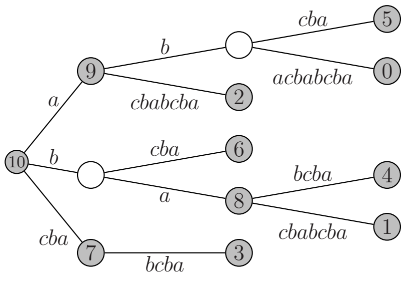
$$c(\mathbf{q}) = (q_0^R, q_0^S, q_0^R, q_1^S, q_1^R, q_2^S, q_2^R, \ldots).$$

We end this section by mentioning the following well-known result (see [3])

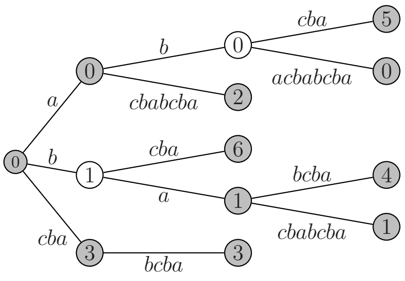
Proposition 1. *The Ziv-Lempel an the Crochemore factorizations of a finite word* x *can be computed in linear time.*

Indeed, one first computes the suffix tree of the word x, where each final state is labelled with the position of its suffix in x (see Figure 1(a)). Then, one computes, fore each vertex, the smallest of all positions of the factor corresponding to this vertex. This is done in linear time by a bottom up tree traversal to compute the minimum of all positions of its descendants (see Figure 1(b)).

To compute the Crochemore factorization $c(x) = (x_1, \ldots, x_n)$ of x, assume it is computed up to x_j. One enters the suffix $y = x_j \cdots x_n$ into the suffix tree as far as possible, provided the position red in the suffix tree remains strictly smaller than $|x_1 \cdots x_{j-1}|$. The maximal prefix of y obtained is x_j. This algorithm is clearly linear.



(a) The suffix tree of *abacbabcba*



(b) The augmented suffix tree

Fig. 1. The suffix tree and the extended suffix tree of the word *abacbabcba*

3 Crochemore Factorization of Standard Sturmian Words

We recall that a standard Sturmian word is the limit $\mathbf{s}$ of the sequence

$$s_{-1} = b, s_0 = a, \text{ and } s_n = s_{n-1}^{d_n} s_{n-2},$$

with $\{d_n\}_{n>0}$ a sequence of positive integers. We want now to prove Theorem 1. It is a well known fact that $s_n = p_n \varepsilon_n$ for each n, where p_n is the palindrome word obtained by deleting the last two letters ε_n of s_n, and

$$\varepsilon_n = \begin{cases} ab & \text{if } n \text{ is odd} \\ ba & \text{otherwise.} \end{cases}$$

With an easy induction argument, one can obtain the following result.

Proposition 2. $s_n = s_{n-1}^{d_n - 1} s_{n-2}^{d_{n-1}} \cdots s_0^{d_1} \varepsilon_n$ *for each* $n > 0$.

Since $s_n = p_n \varepsilon_n$ and p_n is a palindrome word, one immediately has the following decomposition of $\mathbf{s}$ in reverse words.

Proposition 3. $\mathbf{s} = \widetilde{s_0}^{\,d_1} \widetilde{s_1}^{\,d_2} \widetilde{s_2}^{\,d_3} \cdots$.

Lemma 1. *Let* w_n *be the word* $s_n^{d_{n+1}} s_{n-1}^{d_n} s_{n-2}^{d_{n-1}} \cdots s_0^{d_1}$. *Then the only occurrences of* s_n *in* w_n *are the first* $d_{n+1} + 1$ *consecutive ones.*

Proof. Since s_n is primitive for every n, s_n is not a proper factor of s_n^2 and we only have to prove that the $(d_{n+1} + 1)$-th occurrence is the last one. We prove it by induction on n. If $n = 2$, $s_2 = (a^{d_1} b)^{d_2} a$ and $w_2 = s_2^{d_3} w_1 = ((a^{d_1} b)^{d_2} a)^{d_3} (a^{d_1} b)^{d_2} a^{d_1}$. Since $(a^{d_1} b)^{d_2}$ occurs in w_1 only as a prefix, we have the assertion. If $n = 3$, $s_3 = ((a^{d_1} b)^{d_2} a)^{d_3} a^{d_1} b$ and $w_3 = s_3^{d_4} w_2$. Since the last occurrence of $aa^{d_1} b$, which is a suffix of s_3, in w_3 is exactly the suffix of the initial $s_2^{d_3} s_1$ in w_2, we have the assertion. Let now suppose the assertion be true for every $2 \le k < n$. $w_n = s_n^{d_n + 1} w_{n-1}$. By induction hypothesis, there are only two occurrences of $s_{n-1}^{d_n}$ in w_{n-1} and only the first of them is followed by s_{n-2}, so that s_n occurs in w_{n-1} only as a prefix. □

It is now easy to prove Theorem 1.

Proof. **(Theorem 1)** Consider $\mathbf{s}$ as a composition of reverse words as obtained in Proposition 3. The first 4 factors of the c-factorization can be easily obtained by hand. Notice that the composition of these 4 factors in the c-factorization is exactly $\widetilde{s_0}^{d_1}\widetilde{s_1}^{d_2}$:

$$c(\mathbf{s}) = (\underbrace{a, a^{d_1-1}, b, \widetilde{s}_0^{d_1}\widetilde{s}_1^{d_2-1}}_{\widetilde{s_0}^{d_1}\widetilde{s_1}^{d_2}}, \widetilde{s}_2^{d_3}, \widetilde{s}_3^{d_4}, \ldots).$$

We obtain the result as the limit of the c-factorization on the palindrome prefixes p_n of $\mathbf{s}$. Suppose the c-factorization of $p_n = \widetilde{s}_0^{d_1} \cdots \widetilde{s}_{n-2}^{d_{n-1}}\widetilde{s}_{n-1}^{d_n-1}$ to be

$$c(p_n) = (\ldots, \widetilde{s}_{n-2}^{d_{n-1}}, \widetilde{s}_{n-1}^{d_n-1}).$$

By Lemma 1, the only occurrences of $\widetilde{s}_n$ in $\widetilde{w}_n$ are the last $d_{n+1}+1$ consecutive ones and since the first letter of $\widetilde{s}_{n-1}$ is different from the first letter of $\widetilde{s}_n$, we obtain the c-factorization of $p_{n+1} = \widetilde{w}_{n-1}\widetilde{s}_{n-1}^{d_n}\widetilde{s}_n^{d_{n+1}-1}$

$$c(p_{n+1}) = (\ldots, \widetilde{s}_{n-2}^{d_{n-1}}, \widetilde{s}_{n-1}^{d_n}, \widetilde{s}_n^{d_{n+1}-1}). \qquad \square$$

4 Crochemore Factorization of Thue-Morse and Period Doubling Sequences

Let $\mathbf{t}$ be the Thue-Morse word defined as the limit of the morphism τ such that $\tau(a) = ab$ and $\tau(b) = ba$.
We recall the following well known fact.

Lemma 2. *Let w be a factor of $\mathbf{t}$ such that $|w| \geq 4$. Then the occurrences of w in $\mathbf{t}$ begin all in pair positions or all in odd positions.*

Proof. **(Theorem 2)** Let $c(\mathbf{t}) = (c_1, c_2, \ldots)$ be the c-factorization of $\mathbf{t}$. We will prove that each Crochemore factor c_{n+1} begins in a pair position, double than that of c_{n-1}, and that $c_{n+1} = \tau(c_{n-1})$ for every $n \geq 8$. One can verify by hands the first step, that is, c_9 begins in the pair position double than that of c_7. Let now c_{n+1} begin in the double position than that of c_{n-1}. Then $\tau(c_{n-1}) \in \text{Prefix}(c_{n+1}c_{n+2}\cdots)$. By definition of Crochemore factor and by Lemma 2, since $|c_{n-1}| > 4$ the factor c_{n-1} occurs earlier in a pair position, so that also $\tau(c_{n-1})$ has an earlier occurrence. Thus, $\tau(c_{n-1}) \in \text{Prefix}(c_{n+1})$.

By contradiction, suppose $\tau(c_{n-1})a \in \text{Prefix}(c_{n+1})$, where by definition of τ a is forced to be the first letter of c_n. Then there is an earlier occurrence of $\tau(c_{n-1})a$ in a pair position, so that also c_{n-1} occurs followed by a letter a earlier than as a Crochemore factor, that is absurd. $\square$

Let now $\mathbf{t^{(m)}}$ be the generalized Thue-Morse word on the m-alphabet $A = \{a_1, \ldots, a_m\}$ defined as the limit of the morphism

$$\tau_m(a_i) = a_i a_{i+1} \cdots a_m a_1 \cdots a_{i-1} \quad (i = 1, \ldots, m).$$

Set $w_{n,i} = \tau_m^n(a_i)$ and let $t_n^{(m)} = \tau_m^n(a_1) = w_{n,1}$. The following result can be easily obtained by induction.

Lemma 3. *$w_{n,i} \notin Fact(w_{n,h}w_{n,k})$ for every i and $h \neq k$.*

The relationship between the structure of the generalized Thue-Morse words and the c-factorization is simpler than what we have obtained for the Thue-Morse word on a 2-letter alphabet. For example, let $m = 3$. Then

$$c(t_3^{(3)}) = (a, b, c, bc, a, ca, b, bcacab, abc, cababc, bca).$$

Using Lemma 3 it is not difficult to prove the following result, which is even stronger than what we stated in Theorem 3.

Theorem 5. *The c-factorization of $t_n^{(m)}$ is $(w_{1,1}, \dots, w_{1,m}, w_{1,2} \cdots w_{1,m}, w_{1,1}, \dots, w_{1,m} \cdots w_{1,1}, w_{1,m-1}, \dots, w_{n,2} \cdots w_{n,m}, w_{n,1}, \dots, w_{n,m} \cdots w_{n,1}, w_{n,m-1})$.*

In the case $m = 3$ we have

$$t_3^{(3)} = \overbrace{\underbrace{abc}_{w_{2,1}}\,\underbrace{bca}_{w_{2,2}}\,\underbrace{cab}_{w_{2,3}}}^{t_2^{(m)}}\ \underbrace{bcacababc}_{w_{2,2}w_{2,3}w_{2,1}}\ \underbrace{cababcbca}_{w_{2,3}w_{2,1}w_{2,2}},$$

in accordance with the c-factorization given above.

In the general case also, this factorization reflects exactly the decomposition of each prefix $t_n^{(m)}$ in $w_{h,i}$ words:

$$(\underbrace{\underbrace{w_{1,1}, \dots, w_{1,m}}_{t_1^{(m)}=w_{2,1}}, \underbrace{w_{1,2} \cdots w_{1,m}, w_{1,1}}_{w_{2,2}}, \dots, \underbrace{w_{1,m} \cdots w_{1,1}, w_{1,m-1}}_{w_{2,m}}}_{t_2^{(m)}}, \dots)$$

Recall that the period doubling sequence is defined as the limit $\mathbf{q}$ of the morphism δ such that $\delta(a) = ab$ and $\delta(b) = aa$. In order to prove Theorem 4, we begin by providing some easy to prove results on the structure of $\mathbf{q}$.

Lemma 4. *The following facts hold:*

(i) $q_{n+1} = q_n q'_n$, *where if* $q_n = va$, $q'_n = v\bar{a}$.
(ii) $q_n = p_n u_n$, *where* p_n *is a palindrome word such that* $p_0 = \varepsilon$, $p_{n+1} = p_n u_n p_n$, *and*
$$u_n = \begin{cases} a & n \text{ pair} \\ b & n \text{ odd} \end{cases}$$
(iii) $p_n = q_0^R q_1^R \cdots q_{n-1}^R$.

Lemma 5. *For every $n \geq 1$, p_n occurs only as a prefix and as a suffix in p_{n+1}.*

Proof. True for $n = 1, 2$. Suppose the lemma true for $n \leq k$.

$$p_{k+2} = p_k u_k p_k u_{k+1} p_k u_k p_k$$

and $p_{k+1} = p_k u_k p_k$. Since p_{k+1} is different from the central factor of p_{k+2} $p_k u_{k+1} p_k$ and by the induction hypothesis on p_k, we have the assert. □

Proof. **(Theorem 4)** We prove that

$$c(q_{n+2}) = (q_0^R, q_0^S, q_0^R, q_1^S, q_1^R, \ldots, q_n^S, q_n^R, u(n)).$$

The assertion holds for $n = 2$. Suppose it is true for q_{n+1}.
By Lemma 4,

$$q_{n+2} = q_{n+1} q'_{n+1} = p_{n+1} u_{n+1} p_n u_n p_n u_n. \tag{1}$$

Note that $u_{n+1} p_n = q_n^S$ and $u_n p_n = q_n^R$.
By induction hypothesis, the last Crochemore factor of q_{n+1} is its last letter, so that the u_{n+1} of Equation (1) is the first letter of a Crochemore factor of q_{n+2}.
With the further expansion of the expression of q_{n+2}

$$q_{n+2} = p_{n-1} u_{n-1} p_{n-1} u_n p_{n-1} u_{n-1} p_{n-1} \underbrace{u_{n-1} p_{n-1} u_{n-1} p_{n-1}}_{q_n^S} \underbrace{u_n p_{n-1} u_{n-1} p_{n-1}}_{q_n^R} u_n$$

we can verify that the factors $u_{n+1} p_n$ and $u_n p_n$ of Equation (1) (here underlined in the same occurrences) occurred already before in q_{n+2}. Moreover, by using Lemma 5 it can be shown that they are not contained into larger already occurred factors, that is, $u_{n+1} p_n u_n$ and $u_n p_n u_n$ did not occur before, so that q_n^S, q_n^R, u_n are exactly the c-factors we need to add to those of q_{n+1} to complete the c-factorization of q_{n+2}. □

5 Crochemore Factorization Versus Ziv-Lempel Factorization

Ziv and Lempel have considered several variations of factorizations of words (see [7]; these are also discussed in [6]). We illustrate the relation between Crochemore and Ziv-Lempel factorizations by stating some simple facts, and by giving some examples.

Lemma 6. *Let* $(c_1, c_2, \ldots)$ *and* $(z_1, z_2, \ldots)$ *be the Crochemore and the Ziv-Lempel factorizations of a word* w. *The following hold.*

- *For each* i, j *such that* $|c_1 \cdots c_{i-1}| \geq |z_1 \cdots z_{j-1}|$ *and* $|c_1 \cdots c_i| < |z_1 \cdots z_j|$, *then* $|z_1 \cdots z_j| = |c_1 \cdots c_i| + 1$.
- *For each* i, j *such that* $|z_1 \cdots z_{j-1}| < |c_1 \cdots c_i| \leq |z_1 \cdots z_j|$ *then* $|c_1 \cdots c_{i+1}| \leq |z_1 \cdots z_{j+1}|$.

Lemma 6 reflects the fact that by their definitions if a Ziv-Lempel factor includes a Crochemore factor, then it ends at most a letter after, and that a Crochemore factor cannot include a Ziv-Lempel factor. Accordingly, we have the following result.

Proposition 4. *The number of factors of the Crochemore factorization is at most twice the number of factors of the Ziv-Lempel factorization.*

Consider for example the period doubling sequence. It is simple to show that each Ziv-Lempel factor of $\mathbf{q}$ properly includes a Crochemore factor by ending just a letter before, as illustrated in this figure:

$$\begin{array}{rl} z: & \mid a \mid b \mid a\ a \mid a\ b\ a\ b \mid a\ b\ a\ a\ a\ b\ a\ a \mid \ \cdots \\ c: & \mid a \mid b \mid a \mid a\ a \mid b\ a \mid b\ a\ b\ a \mid a\ a\ b\ a \mid a\ \ \cdots \end{array}$$

In this example each Ziv-Lempel can therefore be associated to a couple of Crochemore factors. On the contrary, in the next example each Ziv-Lempel factor is associated to a sole factor.

Consider the word $\mathbf{v} = aabbabbbbabbbbbba \cdots$ in which for each $i \geq 0$ the letter in position i is defined as a if i is a perfect square and b otherwise. The Ziv-Lempel factors of $\mathbf{v}$ are shifted one letter ahead with respect to the Crochemore factors, as illustrated in the figure:

$$\begin{array}{rl} z: & \mid a \mid a\ b \mid b\ a \mid b\ b\ b \mid b\ a\ b\ b\ b\ b\ b \mid b\ a\ \ \cdots \\ c: & \mid a \mid a \mid b \mid b \mid a\ b\ b \mid b\ b\ a\ b\ b\ b\ b \mid b\ b\ a\ \ \cdots \end{array}$$

6 Conclusion

In the examples given here, the detailed knowledge of the structure of the infinite words yields enough information in order to compute the Crochemore factorization. Similar results hold for episturmian words. On the contrary, it is not yet clear whether a satisfactory description can be obtained for automatic sequences other than those which are uniform purely morphic sequences.

References

1. J.-P. Allouche and J. Shallit, *Automatic Sequences*, Cambridge University Press, 2003.
2. M. Crochemore, Recherche linéaire d'un carré dans un mot, *Comptes Rendus Sci. Paris Sér. I Math.*, 1983, **296**, 781–784.
3. M. Crochemore, C. Hancart, and T. Lecroq, *Algorithmique du texte*, Vuibert, 2001.
4. M. Crochemore and W. Rytter, *Text Algorithms*, The Clarendon Press Oxford University Press, 1994.
5. A. de Luca, A division property of the Fibonacci word, *Information Processing Letters*, 1995, **54**, 307–312.
6. R. Kolpakov and G. Kucherov, Periodic structures on words, in Lothaire, *Applied Combinatorics on Words*, Cambridge University Press, 2005.
7. A. Lempel and J. Ziv, On the complexity of finite sequences, *IEEE Transactions in Information Theory*, 1976, **IT-22**, 75–81.
8. Z.-X. Wen and Z.-Y. Wen, Some properties of the singular words of the Fibonacci word. *European Journal of Combinatorics* 1994, **15**, 587–598.

Equations on Partial Words

F. Blanchet-Sadri*, D. Dakota Blair, and Rebeca V. Lewis

Department of Mathematical Sciences, University of North Carolina,
P.O. Box 26170, Greensboro, NC 27402–6170, USA

Abstract. It is well known that some of the most basic properties of words, like the commutativity ($xy = yx$) and the conjugacy ($xz = zy$), can be expressed as solutions of word equations. An important problem is to decide whether or not a given equation on words has a solution. For instance, the equation $x^m y^n = z^p$ has only periodic solutions in a free monoid, that is, if $x^m y^n = z^p$ holds with integers $m, n, p \geq 2$, then there exists a word w such that x, y, z are powers of w. This result, which received a lot of attention, was first proved by Lyndon and Schützenberger for free groups. In this paper, we investigate equations on *partial words*. Partial words are sequences over a finite alphabet that may contain a number of "do not know" symbols. When we speak about equations on partial words, we replace the notion of equality ($=$) with compatibility ($\uparrow$). Among other equations, we solve $xy \uparrow yx$, $xz \uparrow zy$, and special cases of $x^m y^n \uparrow z^p$ for integers $m, n, p \geq 2$. ...

1 Introduction

An important topic in algorithmic combinatorics on words is the *satisfiability problem for equations on words*, that is, the problem to decide whether or not a given equation on the free monoid has a solution. The problem was proposed in 1954 by Markov [16] and remained open until 1977 when Makanin answered it positively [15]. However, Makanin's algorithm is one of the most complicated algorithms ever presented and has at least exponential space complexity [13]. Rather recently, Plandowski showed, with a completely new algorithm, that the problem is actually in polynomial space [17,18]. However, the structure of the solutions cannot be found using Makanin's algorithm. Even for rather short instances of equations, for which the existence of solutions may be easily established, the structure of the solutions may be very difficult to describe.

It is well known that some of the most basic properties of words, like the *commutativity* and the *conjugacy* properties, can be expressed as solutions of word equations. Two words x and y commute, namely $xy = yx$, if and only if x

* This material is based upon work supported by the National Science Foundation under Grant No. DMS–0452020. We thank Margaret Moorefield for very valuable help in the implementation of the programs and the creation of the World Wide Web site at `http://www.uncg.edu/mat/research/equations/` for this research. We thank the referees of a preliminary version of this paper for their very valuable comments and suggestions.

and y are powers of the same word, that is, there exists a word z such that $x = z^m$ and $y = z^n$ for some integers m and n. Two words x and y are *conjugate* if there exist words v and w such that $x = vw$ and $y = wv$. The latter is equivalent to the existence of a word z satisfying $xz = zy$ in which case there exist words v, w such that $x = vw$, $y = wv$, and $z = (vw)^n v$ for some nonnegative integer n. The equation $x^m y^n = z^p$ has only periodic solutions in a free semigroup, that is, if $x^m y^n = z^p$ holds with integers $m, n, p \geq 2$, then there exists a word w such that x, y, z are powers of w. This result, which received a lot of attention, was first proved by Lyndon and Schützenberger for free groups [14]. Their proof implied the case for free semigroups since every free semigroup can be embedded in a free group. Direct proofs for free semigroups appear in [9,10,12].

In this paper, we investigate *equations on partial words.* When we speak about them, we replace the notion of *equality* with the notion of *compatibility.* A fundamental difference between equality and compatibility is that the latter is not transitive which makes this paper's results on partial words nontrivial adaptations of the corresponding results on words. Reference [11] presents some motivation from molecular biology for studying this type of equations on partial words. The contents of our paper are summarized as follows: Section 2 is devoted to reviewing basic concepts on words and partial words. There, we define in particular the containment relation ($\subset$) and the compatibility relation ($\uparrow$) on partial words. In Section 3, we give a result that expounds on the idea of the specialty of partial words satisfying the equation $x^m \uparrow y^n$. This result provides motivation on the conditions for when x and y are contained in powers of a common word. Section 4 reviews results on the equation $xy \uparrow yx$ on partial words that will be needed in later sections of our paper. In Section 5, we investigate the conjugacy equation $xz \uparrow zy$ on partial words. Our result is based on an algorithm that decomposes a weakly $|x|$-periodic partial word z satisfying $xz \uparrow zy$ into a product of subwords v_i and w_i pairs such that $|v_i| = |z| \bmod |x|$ and $|w_i| = |x| - |v_i|$. We also study the system of equations $z \uparrow z'$ and $xz \uparrow z'y$. If $z = z'$, then this implies $xz \uparrow zy$. In Section 6, the equation $x^2 \uparrow y^m z$ on partial words is solved. This result is a first step for studying the equation $x^m y^n \uparrow z^p$ discussed in Section 7.

2 Preliminaries

Herein lies a brief description of terms and notations used for words and partial words.

Let A be a nonempty finite set of symbols called an *alphabet.* Symbols in A are called *letters* and any finite sequence over A is called a *word* over A. The *empty word*, that is the word containing no letter, is denoted by ϵ. For any word u over A, $|u|$ denotes the number of letters occurring in u and is called the *length* of u. In particular, $|\epsilon| = 0$. The set of all words over A is denoted by A^*. If we define the operation of two words u and v of A^* by juxtaposition (or concatenation), then A^* is a monoid with identity ϵ. We call $A^+ = A^* \setminus \{\epsilon\}$ the *free semigroup generated by* A and A^* *the free monoid generated by* A. The set A^* can also be viewed as $\bigcup_{n \geq 0} A^n$ where $A^0 = \{\epsilon\}$ and A^n is the set of all words of length n over A.

A word of length n over A can be defined by a total function $u : \{0, \ldots, n-1\} \to A$ and is usually represented as $u = a_0 a_1 \ldots a_{n-1}$ with $a_i \in A$. A *period* of u is a positive integer p such that $a_i = a_{i+p}$ for $0 \leq i < n - p$. For a word u, the powers of u are defined inductively by $u^0 = \epsilon$ and, for any $i \geq 1$, $u^i = uu^{i-1}$. The *reversal* of u, denoted by $rev(u)$, is defined as follows: If $u = \epsilon$, then $rev(\epsilon) = \epsilon$, and if $u = a_0 a_1 \ldots a_{n-1}$, then $rev(u) = a_{n-1} \ldots a_1 a_0$. A word u is a *factor* of the word v if there exist words x, y such that $v = xuy$. The factor u is called *proper* if $u \neq \epsilon$ and $u \neq v$. The word u is a *prefix* (respectively, *suffix*) of v if $x = \epsilon$ (respectively, $y = \epsilon$). A nonempty word u is *primitive* if there exists no word v such that $u = v^n$ with $n \geq 2$. Note the fact that the empty word is not primitive. If u is a nonempty word, then there exist a unique primitive word v and a unique positive integer n such that $u = v^n$.

Now, a *partial word* u of length n over A is a partial function $u : \{0, \ldots, n-1\} \to A$. For $0 \leq i < n$, if $u(i)$ is defined, then we say that i belongs to the *domain* of u, denoted by $i \in D(u)$, otherwise we say that i belongs to the *set of holes* of u, denoted by $i \in H(u)$. A word over A is a partial word over A with an empty set of holes (we sometimes refer to words as *full* words).

If u is a partial word of length n over A, then the *companion* of u denoted by $u_\diamond$, is the total function $u_\diamond : \{0, \ldots, n-1\} \to A \cup \{\diamond\}$ defined by $u_\diamond(i) = u(i)$ if $i \in D(u)$, and $\diamond$ otherwise. The bijectivity of the map $u \mapsto u_\diamond$ allows us to define for partial words concepts such as concatenation, powers, reversals, factors, prefixes, suffixes, etc... in a trivial way. For instance, the reversal of u is defined by $(rev(u))_\diamond = rev(u_\diamond)$. The character $\diamond \notin A$ is viewed as a "do not know" character. The word $u_\diamond = abb\diamond bbcbb$ is the companion of the partial word u of length 9 where $D(u) = \{0, 1, 2, 4, 5, 6, 7, 8\}$ and $H(u) = \{3\}$. The length of the companion of a partial word u, also called the length of u, is denoted by $|u|$, and the set of distinct letters in A occurring in $u_\diamond$ is denoted by $\alpha(u)$. The set of all partial words over A with an arbitrary number of holes is denoted by $W(A)$. It is a monoid under the operation of concatenation with identity ϵ.

A *period* of a partial word u is a positive integer p such that $u(i) = u(j)$ whenever $i, j \in D(u)$ and $i \equiv j \bmod p$. In this case, we call u *p-periodic*. The smallest period of u is called the *minimal period* of u and is denoted by $p(u)$. A *weak period* of u is a positive integer p such that $u(i) = u(i + p)$ whenever $i, i+p \in D(u)$. In this case, we call u *weakly p-periodic*. The smallest weak period of u is called the *minimal weak period* of u and is denoted by $p'(u)$. Note that every weakly p-periodic full word is p-periodic but this is not necessarily true for partial words. Also even if the length of a partial word u is a multiple of a weak period of u, then u is not necessarily a power of a shorter partial word.

If u and v are partial words of equal length, then u is said to be contained in v denoted by $u \subset v$, if all symbols in $D(u)$ are in $D(v)$ and $u(i) = v(i)$ for all $i \in D(u)$. A partial word u is *primitive* if there exists no word v such that $u \subset v^n$ with $n \geq 2$. Note that if v is primitive and $v \subset u$, then u is primitive as well. It was shown in [2] that if u is a nonempty partial word, then there exist a primitive word v and a positive integer n such that $u \subset v^n$. However uniqueness does not hold as seen with the partial word u where $u_\diamond = \diamond a$ (here $u \subset a^2$ and

$u \subset ba$ for distinct letters a, b). There, it was also shown that for partial words u and v, if there exists a primitive word x such that $uv \subset x^n$ for some positive integer n, then there exists a primitive word y such that $vu \subset y^n$. Moreover, if uv is primitive, then vu is primitive. These results extend similar results for words [19]. Also, it is immediate that if u is a primitive partial word, then $rev(u)$ is also primitive.

The partial words u and v are called *compatible*, denoted by $u \uparrow v$, if there exists a partial word w such that $u \subset w$ and $v \subset w$. We denote by $u \vee v$ the least upper bound of u and v. In other words, $u \subset u \vee v$ and $v \subset u \vee v$ and $D(u \vee v) = D(u) \cup D(v)$. As an example, $u_\diamond = aba\diamond\diamond a$ and $v_\diamond = a\diamond\diamond b\diamond a$ are the companions of two partial words u and v that are compatible and $(u \vee v)_\diamond = abab\diamond a$. The following rules are useful for computing with partial words [1]: *Multiplication:* If $u \uparrow v$ and $x \uparrow y$, then $ux \uparrow vy$; *Simplification:* If $ux \uparrow vy$ and $|u| = |v|$, then $u \uparrow v$ and $x \uparrow y$; and *Weakening:* If $u \uparrow v$ and $w \subset u$, then $w \uparrow v$. For convenience, we will refer to a partial word over A as a word over the enlarged alphabet $A \cup \{\diamond\}$, where the additional symbol $\diamond$ plays a special role. This allows us to say for example "the partial word $ab\diamond a\diamond b$" instead of "the partial word with companion $ab\diamond a\diamond b$".

3 The Equation $x^m \uparrow y^n$ on Partial Words

In this section, we investigate the equation $x^m \uparrow y^n$ on partial words. The equation $x^m = y^n$ on words is well known. Indeed, if x and y are words, then $x^m = y^n$ for some positive integers m, n if and only if there exists a word z such that $x = z^k$ and $y = z^l$ for some integers k, l. When dealing with partial words x and y, if there exists a partial word z such that $x \subset z^k$ and $y \subset z^l$ for some integers k, l, then $x^m \uparrow y^n$ for some positive integers m, n. Indeed, by the multiplication rule, $x^l \subset z^{kl}$ and $y^k \subset z^{kl}$, showing that $x^l \uparrow y^k$. For the converse, it is beneficial to define the following manipulation of a partial word x. For a positive integer p and an integer $0 \le i < p$, define $x\begin{bmatrix} i \\ p \end{bmatrix}$ as

$$x(i)x(i+p)x(i+2p)\dots x(i+jp)$$

where j is the largest nonnegative integer such that $i + jp < |x|$. We shall call this the ith residual word of x modulo p.

Using the multiplication and the simplification rules, we can demonstrate that if x, y are partial words and m, n and p are positive integers, then $x^m \uparrow y^n$ if and only if $x^{mp} \uparrow y^{np}$. Consequently, if $x^{m'} \uparrow y^{n'}$ and $\gcd(m', n') \neq 1$, then $x^m \uparrow y^n$ where $m = m'/\gcd(m', n')$ and $n = n'/\gcd(m', n')$. And therefore the assumption that $\gcd(m, n) = 1$ may be made without losing generality.

Lemma 1. *Let x, y be partial words and let m, n be positive integers such that $x^m \uparrow y^n$ with $\gcd(m, n) = 1$. Call $|x|/n = |y|/m = p$. If there exists an integer i such that $0 \le i < p$ and $x\begin{bmatrix} i \\ p \end{bmatrix}$ is not 1-periodic, then $D(y\begin{bmatrix} i \\ p \end{bmatrix})$ is empty.*

Lemma 2. *Let x be a partial word, let m, p be positive integers, and let i be an integer such that $0 \leq i < p$. Then the relation*

$$x^m\begin{bmatrix} i \\ p \end{bmatrix} = x\begin{bmatrix} i \\ p \end{bmatrix} \; x\begin{bmatrix} (i - |x|) \bmod p \\ p \end{bmatrix} \; x\begin{bmatrix} (i - 2|x|) \bmod p \\ p \end{bmatrix} \cdots \; x\begin{bmatrix} (i - (m-1)|x|) \bmod p \\ p \end{bmatrix}$$

holds.

Theorem 1. Good pairs

Let x, y be partial words and let m, n be positive integers such that $x^m \uparrow y^n$ with $\gcd(m, n) = 1$. Assume that for all $i \in H(x)$ the word $y^n\begin{bmatrix} i \\ |x| \end{bmatrix}$ is 1-periodic and that for all $i \in H(y)$ the word $x^m\begin{bmatrix} i \\ |y| \end{bmatrix}$ is 1-periodic (a pair of partial words (x, y) which satisfies this property we will refer to as a "good pair"). Then there exists a partial word z such that $x \subset z^k$ and $y \subset z^l$ for some integers k, l.

Proof. Since $\gcd(m, n) = 1$, there exists an integer p such that $\frac{|x|}{n} = \frac{|y|}{m} = p$. Now assume there exists an integer i such that $0 \leq i < p$ and $x\begin{bmatrix} i \\ p \end{bmatrix}$ is not 1-periodic. Then by Lemma 1, $i + jp \in H(y)$ for $0 \leq j < m$ which by the assumption that (x, y) is a good pair implies that $x^m\begin{bmatrix} i + jp \\ |y| \end{bmatrix}$ must be 1-periodic for any choice of j. Note that $|y| = mp$ and similarly $|x| = np$. Therefore by Lemma 2,

$$x^m\begin{bmatrix} i + jp \\ mp \end{bmatrix} = x\begin{bmatrix} i + jp \\ mp \end{bmatrix} \; x\begin{bmatrix} (i + jp - |x|) \bmod mp \\ mp \end{bmatrix} \cdots \; x\begin{bmatrix} (i + jp - (m-1)|x|) \bmod mp \\ mp \end{bmatrix}$$

Clearly $i + jp - l|x| = i + (j - ln)p$ for all l. For $0 \leq j < m$, we claim that $\{(j - ln) \bmod m \mid 0 \leq l < m\} = \{0, 1, \ldots, m-1\}$. Indeed, assuming there exist $0 \leq l_1 < l_2 < m$ such that $(j - l_1 n) \equiv (j - l_2 n) \bmod m$ we get that m divides $(l_1 - l_2)n$, and since $\gcd(m, n) = 1$, that m divides $(l_1 - l_2)$, whence $l_1 = l_2$. So there exist $j_0, j_1, \ldots, j_{m-1}$ such that $j_0 = j$ and $\{j_0, j_1, \ldots, j_{m-1}\} = \{0, 1, \ldots, m-1\}$ and

$$x^m\begin{bmatrix} i + jp \\ mp \end{bmatrix} = x\begin{bmatrix} i + j_0 p \\ mp \end{bmatrix} \; x\begin{bmatrix} i + j_1 p \\ mp \end{bmatrix} \cdots \; x\begin{bmatrix} i + j_{m-1} p \\ mp \end{bmatrix}$$

Since $x^m\begin{bmatrix} i + jp \\ mp \end{bmatrix}$ is 1-periodic, there exists a letter a such that for all $0 \leq k < m$,

$$x\begin{bmatrix} i + j_k p \\ mp \end{bmatrix} \subset a^{m_{j_k}}$$

for some integer m_{j_k}. This contradicts our assumption that there is an i for which $x\begin{bmatrix} i \\ p \end{bmatrix}$ is not 1-periodic (here $x\begin{bmatrix} i \\ p \end{bmatrix} = x(i)x(i+p)\ldots x(i + (n-1)p) \subset a^n$). Therefore $x\begin{bmatrix} i \\ p \end{bmatrix}$ is 1-periodic for all $0 \leq i < p$. By the equivalent condition for periodicity, this implies that x is p-periodic. The same argument holds for y, and since $x^m \uparrow y^n$, the result that there exists a word z of length p such that $x \subset z^n$ and $y \subset z^m$ is proven. $\square$

The example $x^2 = (a\diamond b)^2 \uparrow (acbadb)^1 = y^1$ shows that the assumption of (x, y) being a good pair is necessary in Theorem 1. Here $y(1)y(4) = cd$ is not 1-periodic and there exists no partial word z as desired.

Corollary 1. *Let x and y be primitive partial words such that (x, y) is a good pair. If $x^m \uparrow y^n$ for some positive integers m and n, then $x \uparrow y$.*

Note that if both x and y are full words, then (x, y) is a good pair. Corollary 1 hence implies that if x, y are primitive full words satisfying $x^m = y^n$ for some positive integers m and n, then $x = y$. We conclude this section by further investigating the equation $x^2 \uparrow y^m$ on partial words where m is a positive integer.

Proposition 1. *Let x, y be partial words. Then $x^2 \uparrow y^m$ for some positive integer m if and only if there exist partial words $u, v, u_0, v_0, \ldots, u_{m-1}, v_{m-1}$ such that $y = uv$,*

$$x = (u_0 v_0) \ldots (u_{n-1} v_{n-1}) u_n = v_n (u_{n+1} v_{n+1}) \ldots (u_{m-1} v_{m-1})$$

where $0 \le n < m$, $u \uparrow u_i$ and $v \uparrow v_i$ for all $0 \le i < m$, and where one of the following holds: (1) $m = 2n$ and $u = \epsilon$; or (2) $m = 2n + 1$ and $|u| = |v|$.

4 The Equation $xy \uparrow yx$ Partial Words

It is well known that two nonempty words x and y commute if and only if there exists a word z such that $x = z^m$ and $y = z^n$ for some integers m, n. When dealing with two nonempty partial words x and y, the existence of a word z satisfying $x \subset z^m$ and $y \subset z^n$ for some integers m, n certainly implies $xy \uparrow yx$. To extend the converse to partial words, we first consider xy to have at most one hole.

Theorem 2. Commutativity one hole [1]

Let x and y be nonempty partial words such that xy has at most one hole. If $xy \uparrow yx$, then there exists a word z such that $x \subset z^m$ and $y \subset z^n$ for some integers m, n.

As stated in [1], Theorem 2 is false if xy has two holes. Take for example $x = \diamond bb$ and $y = abb\diamond$. To extend this theorem to the case when xy has at least two holes, we may assume $|x| \le |y|$. The extension is based on the concept of xy not being (k, l)-special where k, l denote the lengths of x, y respectively. For $0 \le i < k + l$, we define the sequence of i relative to k, l as $seq_{k,l}(i) = (i_0, i_1, i_2, \ldots, i_n, i_{n+1})$ where $i_0 = i = i_{n+1}$; where for $1 \le j \le n, i_j \ne i$; and where for $1 \le j \le n + 1, i_j$ is defined as $i_j = i_{j-1} + k$ if $i_{j-1} < l$, and $i_{j-1} - l$ otherwise. For example, if $k = 6$ and $l = 8$, then $seq_{(6,8)}(0) = (0, 6, 12, 4, 10, 2, 8, 0)$. Now, the concept of (k, l)-special is defined as follows.

Definition 1. (k, l)-Special [3]

Let k, l be positive integers satisfying $k \le l$ and let z be a partial word of length $k + l$. We say that z is (k, l)-special if there exists $0 \le i < k$ such that $seq_{k,l}(i) = (i_0, i_1, i_2, \ldots, i_n, i_{n+1})$ contains (at least) two positions that are holes of z while

$$z_\diamond(i_0) z_\diamond(i_1) \ldots z_\diamond(i_{n+1})$$

is not 1-periodic.

If $k = 6$ and $l = 8$, then $z = acbca\diamond\diamond cbc\diamond cac$ is $(6, 8)$-special since $seq_{6,8}(0)$ contains the positions 6 and 10 which are in $H(z) = \{5, 6, 10\}$ while $a\diamond aa\diamond bba$ is not 1-periodic.

Theorem 3. Commutativity arbitrary number of holes [3]
Let x, y be nonempty partial words such that $|x| \leq |y|$. If $xy \uparrow yx$ and xy is not $(|x|, |y|)$-special, then there exists a word z such that $x \subset z^m$ and $y \subset z^n$ for some integers m, n.

The concept of $\{k, l\}$-special and the following two lemmas will be useful in the sequel.

Definition 2. $\{k, l\}$-Special [5]
Let k, l be positive integers satisfying $k \leq l$ and let z be a partial word of length $k + l$. We say that z is $\{k, l\}$-special if there exists $0 \leq i < k$ such that $seq_{k,l}(i)$ satisfies the condition of Definition 1 or the condition of containing two consecutive positions that are holes of z.

If $k = 6$ and $l = 8$, then $z = \diamond babab\diamond\diamond ababab$ is $\{6, 8\}$-special (but is not $(6, 8)$-special). Indeed, $seq_{6,8}(0)$ contains the consecutive positions 0, 6 that are holes of z.

Lemma 3. [1]
Let x, y be nonempty words and let z be a partial word with at most one hole. If $z \subset xy$ and $z \subset yx$, then $xy = yx$.

Lemma 4. [5]
Let x, y be nonempty words and let z be a non $\{|x|, |y|\}$-special partial word. If $z \subset xy$ and $z \subset yx$, then $xy = yx$.

Note that in Lemma 4, the assumption of z being non $\{|x|, |y|\}$-special cannot be replaced by the weaker assumption of z not being $(|x|, |y|)$-special. To see this, consider the partial words $x = ababab$, $y = cbababab$, and $z = \diamond babab\diamond\diamond ababab$. Here, $z \subset xy$ and $z \subset yx$, but $xy \neq yx$.

The concept of (k, l)-special partial word, which relates to commutativity, turned out to be foundational in the design of our linear time algorithm for testing primitivity on partial words [3].

5 The Equation $xz \uparrow zy$ on Partial Words

In this section, we consider the conjugacy property of partial words. Two partial words x and y are conjugate if there exist partial words v and w such that $x \subset vw$ and $y \subset wv$ [5]. It turns out that if the partial words x and y are conjugate, then there exists a partial word z satisfying the conjugacy equation $xz \uparrow zy$. The equation $xz = zy$ on words is well known. Indeed, if z is a word and x, y are nonempty words such that $xz = zy$, then there exist words v, w satisfying $x = vw$, $y = wv$, and $z = (vw)^n v$ for some integer $n \geq 0$. For partial words, the next similar result follows via the assumption of $xz \vee zy$ being $|x|$-periodic.

Theorem 4. [5]
Let x, y, z be partial words with x, y nonempty. If $xz \uparrow zy$ and $xz \vee zy$ is $|x|$-periodic, then there exist words v, w such that $x \subset vw$, $y \subset wv$, and $z \subset (vw)^n v$ for some integer $n \geq 0$.

As noted in [5], if z is a full word, then the assumption $xz \uparrow zy$ implies the one of $xz \vee zy$ being $|x|$-periodic and the following corollary holds. Note that Corollary 2 does not necessarily hold if z is not full even if x, y are full. The partial words $x = a, y = b$, and $z = \diamond bb$ provide a counterexample.

Corollary 2. [5]
Let x, y be nonempty partial words, and let z be a full word. If $xz \uparrow zy$, then there exist words v, w such that $x \subset vw$, $y \subset wv$, and $z \subset (vw)^n v$ for some integer $n \geq 0$.

First, we investigate the equation $xz \uparrow zy$ on partial words under the missing assumption of $xz \vee zy$ being $|x|$-periodic. The following two results give equivalences for conjugacy.

Theorem 5. *Let x, y and z be partial words such that $|x| = |y| > 0$. Then $xz \uparrow zy$ if and only if xzy is weakly $|x|$-periodic.*

Theorem 6. *Let x, y and z be partial words such that $|x| = |y| > 0$. Then the following hold:*

1. *If $xz \uparrow zy$, then xz and zy are weakly $|x|$-periodic.*
2. *If xz and zy are weakly $|x|$-periodic and $\lfloor \frac{|z|}{|x|} \rfloor > 0$, then $xz \uparrow zy$.*

In Theorem 6(2), the assumption $\lfloor \frac{|z|}{|x|} \rfloor > 0$ is necessary. To see this, consider $x = aa$, $y = ba$ and $z = a$. Here, xz and zy are weakly $|x|$-periodic, but $xz \not\uparrow zy$.

Second, we consider solving the system of equations $z \uparrow z'$ and $xz \uparrow z'y$. Note that when $z = z'$, this system reduces to $xz \uparrow zy$. Let m be defined as $\lfloor \frac{|z|}{|x|} \rfloor$ and n as $|z| \bmod |x|$. Then let $x = v_0 w_0$, $y = w_{m+1} v_{m+2}$, $z = v_1 w_1 v_2 w_2 \ldots v_m w_m v_{m+1}$, and $z' = v'_1 w'_1 v'_2 w'_2 \ldots v'_m w'_m v'_{m+1}$ where each v_i, v'_i has length n and each w_i, w'_i has length $|x| - n$. The $|x|$-*pshuffle* and $|x|$-*sshuffle* of xz and $z'y$ are defined as

$$\text{pshuffle}_{|x|}(xz, z'y) = v_0 w_0 v'_1 w'_1 v_1 w_1 v'_2 w'_2 \ldots$$
$$v_{m-1} w_{m-1} v'_m w'_m v_m w_m v'_{m+1} w_{m+1} v_{m+1}$$
$$\text{sshuffle}_{|x|}(xz, z'y) = v_{m+1} v_{m+2}$$

Theorem 7. *Let x, y, z and z' be partial words such that $|x| = |y| > 0$ and $|z| = |z'| > 0$. Then $z \uparrow z'$ and $xz \uparrow z'y$ if and only if pshuffle$_{|x|}(xz, z'y)$ is weakly $|x|$-periodic and sshuffle$_{|x|}(xz, z'y)$ is $(|z| \bmod |x|)$-periodic.*

The results in this section find some nice applications. In [6] for example, Blanchet-Sadri and Wetzler consider one of the most fundamental results on periodicity of words, namely the critical factorization theorem. Given a word w and nonempty words u, v satisfying $w = uv$, the *minimal local period* associated to the factorization (u, v) is the length of the shortest square at position $|u| - 1$. The critical factorization theorem shows that for any word, there is always a factorization whose minimal local period is equal to the minimal period of the word [7,8]. Blanchet-Sadri and Wetzler give a version of the critical factorization theorem for partial words (the one-hole case was considered earlier by Blanchet-Sadri and Duncan [4]. Their proof, which provides an efficient algorithm that computes a critical factorization when one exists, is based on the conjugacy equation on partial words.

6 The Equation $x^2 \uparrow y^m z$ on Partial Words

In this section, we investigate the equation $x^2 \uparrow y^m z$ on partial words where it is assumed that m is a positive integer and z is a prefix of y. This equation has nontrivial solutions (a solution is *trivial* if x, y, z are contained in powers of a common word). Indeed, consider the compatibility relation $(a\diamond\diamond a)^2 \uparrow (aab)^2 aa$ where $x = a\diamond\diamond a$, $y = aab$ and $z = aa$. The equation $x^2 \uparrow y^m z$ will play a crucial role in the study of the equation $x^m y^n \uparrow z^p$ in the next section.

Theorem 8. Good triples
Let x, y, z be partial words such that z is a proper prefix of y. Then $x^2 \uparrow y^m z$ for some positive integer m if and only if there exist partial words $u, v, u_0, v_0, \ldots, u_{m-1}, v_{m-1}, z_x$ such that $u \neq \epsilon$, $v \neq \epsilon$, $y = uv$,

$$x = (u_0 v_0) \ldots (u_{n-1} v_{n-1}) u_n \tag{1}$$
$$= v_n (u_{n+1} v_{n+1}) \ldots (u_{m-1} v_{m-1}) z_x \tag{2}$$

where $0 \leq n < m, u \uparrow u_i$ and $v \uparrow v_i$ for all $0 \leq i < m$, $z \uparrow z_x$, and where one of the following holds:

- *$m = 2n$, $|u| < |v|$, and there exist partial words u', u'_n such that $z_x = u' u_n$, $z = u u'_n$, $u \uparrow u'$ and $u_n \uparrow u'_n$.*
- *$m = 2n + 1$, $|u| > |v|$, and there exist partial words v'_{2n} and z'_x such that $u_n = v_{2n} z_x$, $u = v'_{2n} z'_x$, $v_{2n} \uparrow v'_{2n}$ and $z_x \uparrow z'_x$.*

A triple of partial words (x, y, z) which satisfy these properties we will refer to as a "good triple".

Proof. Note that if the conditions hold, then trivially $x^2 \uparrow y^m z$ for some positive integer m. If $x^2 \uparrow y^m z$ for some positive integer m, then there exist partial words u, v and an integer n such that $y = uv, x \uparrow (uv)^n u$ and $x \uparrow v(uv)^{m-n-1} z$. Thus $|x| = n(|u| + |v|) + |u| = (m - n - 1)(|u| + |v|) + |v| + |z|$ which clearly shows

$$|z| = (2n - m + 2)|u| + (2n - m)|v| \tag{3}$$

This determines a relationship between m and n. There are two cases to consider which correspond to assumptions on $|u|$ and $|v|$. Under the assumption $|u| = |v|$ we see that z must be either empty or equal to y which is a contradiction. If we assume $|u| < |v|$, then (3) shows $|z| = 2|u|$, and if we assume $|u| > |v|$, then $|z| = |u| - |v|$. Now note that x^2 may be factored in the following way:

$$x^2 = (u_0 v_0) \dots (u_{n-1} v_{n-1})(u_n v_n)(u_{n+1} v_{n+1}) \dots (u_{m-1} v_{m-1}) z_x$$

Here $u_i \uparrow u$ and $v_i \uparrow v$ and $z_x \uparrow z$. From this it is clear that (1) and (2) are satisfied.

Note that $u \neq \epsilon$ (otherwise $|u| < |v|$, in which case $|z| = 2|u| = 0$), and also $v \neq \epsilon$ (otherwise, $|u| > |v|$, in which case $|z| = |u| - |v| = |y|$). First assume $|u| < |v|$, equivalently $|z| = 2|u|$ and $m = 2n$. Note that the suffix of length $|u|$ of z_x must be u_n and therefore is compatible with u. The prefix of length $|u|$ of z must be u itself since z is a prefix of y. Thus $z_x = u' u_n$ and $z = u u'_n$ where $u \uparrow u'$ and $u_n \uparrow u'_n$ which is one of our assertions. Now assume $|u| > |v|$, that is $|z| = |u| - |v|$ and $m = 2n + 1$. Note by cancellation that $u_n = v_{2n} z_x$. Since $u_n \uparrow u$, we can rewrite u as $v'_{2n} z'_x$ where $v_{2n} \uparrow v'_{2n}$ and $z_x \uparrow z'_x$, which is our other assertion. □

Corollary 3. *Let x, y be partial words such that $|x| \geq |y| > 0$ and let z be a prefix of y. Assume that $x^2 \uparrow y^m z$ for some positive integer m. Referring to the notation of Theorem 8 (when $z \neq \epsilon$ and $z \neq y$) or referring to the notation of Proposition 1 (otherwise), both $w \uparrow uv$ and $w \uparrow vu$ hold where w denotes the prefix of length $|y|$ of x. Moreover, u and v are contained in powers of a common word if ($z = \epsilon$ and $m = 2n$) or ($z = y$ and $m + 1 = 2n$). This is also true if any of the following six conditions hold with $u \neq \epsilon$ and $v \neq \epsilon$:*

1. *y is full and w has at most one hole.*
2. *y is full and w is not $\{|u|, |v|\}$-special.*
3. *w is full and y has at most one hole.*
4. *w is full, and either ($|u| \leq |v|$ and uv is not $(|u|, |v|)$-special) or ($|v| \leq |u|$ and vu is not $(|v|, |u|)$-special).*
5. *$uv \uparrow vu$ and y has at most one hole.*
6. *$uv \uparrow vu$, and either ($|u| \leq |v|$ and uv is not $(|u|, |v|)$-special) or ($|v| \leq |u|$ and vu is not $(|v|, |u|)$-special).*

Corollary 4. *Let x, y, z be partial words such that z is a prefix of y. Assume that x, y are primitive and that $x^2 \uparrow y^m z$ for some integer $m \geq 2$. If x has at most one hole and y is full, then $x \uparrow y$.*

Corollary 5. [9]

Let x, y, z be words such that z is a prefix of y. If x, y are primitive and $x^2 = y^m z$ for some integer $m \geq 2$, then $x = y$.

Note that Corollaries 4 and 5 do not hold when $m = 1$. Indeed, the words $x = aba$, $y = abaab$ and $z = a$ provide a counterexample. Also, Corollary 4 does not hold when x is full and y has one hole as is seen by setting $x = abaabb$, $y = ab\diamond$ and $z = \epsilon$.

7 The Equation $x^m y^n \uparrow z^p$ on Partial Words

For integers $m \geq 2, n \geq 2$ and $p \geq 2$, Lyndon and Schützenberger showed that the equation $x^m y^n = z^p$ possesses a solution in a free group only when x, y, and z are each a power of a common element [14]. Since every free monoid can be embedded in a free group, the result is true in a free monoid as well (a simpler proof in the case of a free monoid appears in [9]). The equation $x^m y^n \uparrow z^p$ in a free monoid $W(A)$ certainly has a solution when x, y, and z are contained in powers of a common word (we call such solutions the *trivial* solutions). However, there may be nontrivial solutions as is seen with the compatibility relation $(a\diamond b)^2(b\diamond a)^2 \uparrow (abba)^3$. In this section, we characterize some of the solutions of the equation $x^m y^n \uparrow z^p$ for the case where $p \geq 4$. The characterization is stated as Theorem 9 which we show with a series of case proofs. We reduce the number of cases by using the following: If x, y, z are partial words and m, n, p are positive integers satisfying $x^m y^n \uparrow z^p$, then $(rev(y))^n(rev(x))^m \uparrow (rev(z))^p$. It will turn out that, in a free monoid $W(A)$, the equation $x^m y^n \uparrow z^p$, where $m \geq 2, n \geq 2$ and $p \geq 4$, may have solutions of the following types: There exists a partial word w such that x, y, z are contained in powers of w. We call such solutions the *trivial* or *Type 1* solutions; The partial words x, y, z satisfy $x \uparrow z$ and $y \uparrow z$. We call such solutions the *Type 2* solutions. If z is full, then Type 2 solutions are trivial solutions.

Theorem 9. $\boldsymbol{p \geq 4}$
Let x, y, z be primitive partial words such that (x, z) and (y, z) are good pairs. Let m, n, p be integers such that $m \geq 2, n \geq 2$ and $p \geq 4$. Then the equation $x^m y^n \uparrow z^p$ has only solutions of Type 1 or Type 2 unless $x^2 \uparrow z^k z_p$ for some integer $k \geq 2$ and nonempty prefix z_p of z, or $z^2 \uparrow x^l x_p$ for some integer $l \geq 2$ and nonempty prefix x_p of x.

Proof. We need only examine the case when $|x^m| \geq |y^n|$. Now assume $x^m y^n \uparrow z^p$ has some solution that is not of Type 1 or Type 2. Our assumption on the lengths of x^m and y^n implies that $|x^m| \geq |z^2|$ and in any case, either $|x^2| \geq |z^2|$ or $|x^2| < |z^2|$. Hence one of the following equations will be satisfied: $x^2 \uparrow z^k z_p$ for some integer $k \geq 2$ and prefix z_p of z, or $z^2 \uparrow x^l x_p$ for some integer $l \geq 2$ and prefix x_p of x.

Consider the case where z_p or x_p is the empty word. In either case, Corollary 1 implies that $x \uparrow z$. From $x^m y^n \uparrow z^p$ and $x \uparrow z$, we get $y^n \uparrow z^{p-m}$. Using Corollary 1 again, we have $y \uparrow z$. Hence these cases form Type 2 solutions. □

Corollary 6. [9]
Let x, y, z be primitive words and let m, n, p be integers such that $m \geq 2, n \geq 2$ and $p \geq 4$. Then the equation $x^m y^n = z^p$ has no nontrivial solutions.

References

1. Berstel, J., Boasson, L.: Partial Words and a Theorem of Fine and Wilf. Theoret. Comput. Sci. **218** (1999) 135–141
2. Blanchet-Sadri, F.: Primitive Partial Words. Discrete Appl. Math. **148** (2005) 195–213

3. Blanchet-Sadri, F., Anavekar, Arundhati R.: Testing Primitivity on Partial Words. `http://www.uncg.edu/mat/primitive/`
4. Blanchet-Sadri, F., Duncan, S.: Partial Words and the Critical Factorization Theorem. J. Combin. Theory Ser. A **109** (2005) 221–245 `http://www.uncg.edu/mat/cft/`
5. Blanchet-Sadri, F., Luhmann, D.K.: Conjugacy on Partial Words. Theoret. Comput. Sci. **289** (2002) 297–312
6. Blanchet-Sadri, F., Wetzler, N.D.: Partial Words and the Critical Factorization Theorem Revisited. `http://www.uncg.edu/mat/research/cft2/`
7. Césari, Y., Vincent, M.: Une Caractérisation des Mots Périodiques. C.R. Acad. Sci. Paris **268** (1978) 1175–1177
8. Choffrut, C., Karhumäki, J.: Combinatorics of Words. In Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Vol. 1. Springer-Verlag, Berlin (1997) 329–438
9. Chu, D.D., Town, H.S.: Another Proof on a Theorem of Lyndon and Schützenberger in a Free Monoid. Soochow J. Math. **4** (1978) 143–146
10. Harju, T., Nowotka, D.: The Equation $x^i = y^j z^k$ in a Free Semigroup. Semigroup Forum **68** (2004) 488–490
11. Leupold, P.: Partial Words Results and Perspectives. (GRLMC, Tarragona, 2003)
12. Lothaire, M.: Combinatorics on Words. Addison-Wesley, Reading, MA (1983); Cambridge University Press, Cambridge (1997)
13. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
14. Lyndon, R.C., Schützenberger, M.P.: The Equation $a^m = b^n c^p$ in a Free Group. Michigan Math. J. **9** (1962) 289–298
15. Makanin, G.S.: The Problem of Solvability of Equations in a Free Semigroup. *Math. USSR Sbornik* **32** (1977) 129–198
16. Markov, A.A.: The Theory of Algorithms. *Trudy Mat. Inst. Steklov* **42** (1954)
17. Plandowski, W.: Satisfiability of Word Equations with Constants is in NEXPTIME. Annual ACM Symposium on Theory of Computing (Atlanta, GA, 1999) 721–725
18. Plandowski, W.: Satisfiability of Word Equations with Constants is in PSPACE. 40th Annual Symposium on Foundations of Computer Science (New York, 1999) 495–500
19. Shyr, H.J., Thierrin, G.: Disjunctive Languages and Codes. Lecture Notes in Computer Science, Vol. 56. Springer-Verlag, Berlin Heidelberg New York (1977) 171–176

Concrete Multiplicative Complexity of Symmetric Functions

Joan Boyar[1],[⋆] and René Peralta[2],[⋆⋆]

[1] Dept. of Math. and Computer Science, University of Southern Denmark
joan@imada.sdu.dk
[2] Security Division
Information Technology Laboratory, NIST
rene.peralta@nist.gov

Abstract. The multiplicative complexity of a Boolean function f is defined as the minimum number of binary conjunction (AND) gates required to construct a circuit representing f, when only exclusive-or, conjunction and negation gates may be used. This article explores in detail the multiplicative complexity of symmetric Boolean functions. New techniques that allow such exploration are introduced. They are powerful enough to give exact multiplicative complexities for several classes of symmetric functions. In particular, the multiplicative complexity of computing the Hamming weight of n bits is shown to be exactly $n - H^{\mathbb{N}}(n)$, where $H^{\mathbb{N}}(n)$ is the Hamming weight of the binary representation of n. We also show a close relationship between the complexity of symmetric functions and fractals derived from the parity of binomial coefficients.

1 Introduction

Much research in circuit complexity is devoted to the following problem: Given a Boolean function and a supply of gate types, construct a circuit which computes the function and is optimal according to some criteria. It seems to be very difficult in general to obtain exact bounds for specific functions. The *multiplicative complexity* $c_\wedge(f)$ of a Boolean function f is the number of conjunctions necessary and sufficient to implement a circuit which computes f over the basis $(\wedge, \oplus, 1)$ (alternatively, the number of multiplications necessary and sufficient to calculate a function over GF_2 via a straight-line program).

Our initial motivation for studying multiplicative complexity came from cryptography. Many cryptographic protocols involve proving predicates about a string X that is available in *committed* form only, i.e., the bits of X are individually encrypted using a *bit-commitment scheme*. In [3] a construction is given for a non-interactive cryptographic proof of an arbitrary predicate F on X. The predicate F is defined by a verification circuit C containing AND, NOT, and XOR gates only. The length of these

[⋆] Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT), and by the Danish Natural Science Research Council (SNF).

[⋆⋆] Part of this work was done at the Computer Science Department, Yale University, prior to this author joining NIST. While at Yale, this work was partially supported by NSF grant CCR-0081823.

discreet proofs is linear in the number of AND gates in C and is unaffected by the number of NOT or XOR gates. Another promising area of application of these results is in the communication complexity of secure multi-party computation. In general, for these protocols, multiplications require communication, but linear operations do not. This holds for very different paradigms for building protocols, those based on secret sharing were introduced in [2,7] and those based on threshold homomorphic encryption were introduced in [6]. For more recent results, see [9].

We focus on symmetric functions, which are functions dependent only on the Hamming weight $\overrightarrow{H}(\mathbf{x})$ of the input $\mathbf{x} \in GF_2^n$. Obtaining tight bounds is important because symmetric functions can be building blocks for arithmetic circuits, some of which involve recursive use of simple symmetric functions. Sub-optimal implementations of the latter, even by an additive constant factor, translate into multiplicative extra costs when building arithmetic circuits. In cryptographic applications, whether or not a circuit is of practical use often depends on constant multiplicative factors in the number of AND gates used.

The study of multiplicative complexity may prove useful in obtaining upper bounds on the computational complexity of functions. If a function f has multiplicative complexity $O(\log(n))$, then, for all $\mathbf{x}$ in the domain of f, an element of the pre-image of $y = f(\mathbf{x})$ can be found in polynomial-time as follows: Guess the values of inputs to the AND gates in a circuit for f. This reduces the circuit to a collection of linear circuits. Now find an $\mathbf{x}$ such that $y = f(\mathbf{x})$ using Gaussian elimination over GF_2. This shows that, one-way functions, if they exist, have super-logarithmic multiplicative complexity. On the other hand, low multiplicative complexity circuits may lead to better algorithms for inverting functions of importance in cryptology.

Previous work. Multiplicative complexity has been investigated previously by Aleksanyan [1], Schnorr [14], and Mirwald and Schnorr [11]. Their work was exclusively concerned with quadratic forms. Multiplicative complexity has more often been used to refer to more general algebraic computations. This subject has an extensive history (see, for example, [5]), since multiplication is often the dominant operation in this context.

Very little is known about multiplicative complexity of specific functions. In this paper we concentrate on the concrete (as opposed to asymptotic) multiplicative complexity of symmetric functions. In an earlier paper [4], we showed the following results:

– A general upper bound of $n + 3\sqrt{n}$ for any symmetric function f. This establishes a separation between Boolean and multiplicative complexity for symmetric functions. Paul [12] and Stockmeyer [15] have shown lower bounds of the form $2.5n - O(1)$ for the Boolean complexity of infinite families of symmetric functions;
– Let $\Sigma_{.}^{n}$ be the set of symmetric predicates on n bits. We showed an upper bound of $2n - \log_2 n$ for the complexity $c_\wedge(\Sigma_{.}^{n})$ of simultaneously computing all symmetric functions on n bits (the asymptotic result $c_\wedge(\Sigma_{.}^{n}) = O(n)$ was obtained earlier by Mihaĭljuk [10]).

Our results. Several new upper and lower bounds on the multiplicative complexity of symmetric functions are obtained. In particular, it is shown that the multiplicative complexity of computing the Hamming weight is exactly $n - H^{\mathbb{N}}(n)$, where $H^{\mathbb{N}}(n)$ is the Hamming weight of the binary representation of n. This is a rather surprising result, given the sparsity of exact computational complexity bounds known.

A new technique, using a normal form for $(\oplus, 1, \wedge)$ circuits and elementary linear algebra, is used to show that any non-linear symmetric function on n variables has multiplicative complexity at least $\lfloor \frac{n}{2} \rfloor$. Properties of binomial coefficients are shown to yield the following lower bounds for the counting (exactly-k) and threshold-k functions on n variables:

$$c_\wedge(E_k^n) \geq \max\{k-1, n-k-1, 2^{\lfloor \log_2 n \rfloor} - 2, l_{n,k} - 1\}$$
$$c_\wedge(T_k^n) \geq \max\{k-1, n-k, 2^{\lfloor \log_2 n \rfloor} - 1, l_{n-1,k-1}\}$$

where $l_{n,k}$ is the bitwise OR of $n-k$ and k. Tighter bounds for several families of symmetric functions are obtained by considering the multiplicative complexity of such functions when restricted to hyperplanes in GF_2^n. In particular, this technique yields the exact complexities of the elementary symmetric functions $\Sigma_2^n, \Sigma_3^n, \Sigma_{n-1}^n, \Sigma_{n-2}^n, \Sigma_{n-3}^n$. Yet another application of hyperplane restrictions yields new general lower bounds for infinite subclasses of symmetric functions. Intriguingly, these subclasses are defined by fractals on the Cartesian plane.

More constructively, general techniques are developed for proving upper bounds for elementary symmetric functions. These, plus properties of Pascal's triangle modulo 2 (known in the fractals literature as Sierpinski's gasket), are used to prove upper bounds for the counting functions, $E_k^n(\mathbf{x})$, and the threshold functions, $T_k^n(\mathbf{x})$. These general techniques are shown to give many tight results. In addition, a general upper bound on the threshold-k functions, T_k^n, is found: $c_\wedge(T_k^n) \leq n - H^{\mathbb{N}}(n) + \lceil \log_2(n+1) \rceil - 1$ for all $k \geq 1$.

In the following sections, and due to space constraints, most proofs will be omitted.

2 Some Simple Observations and a Normal Form

Each Boolean function f on n variables has a unique representation as a multilinear (i.e. square-free) polynomial over GF_2. Since $x^i = x$ over GF_2, we assume throughout the following that all polynomials are multilinear. By the "degree of f", we will mean the degree of its unique representing polynomial. It is known that a Boolean function of degree d has multiplicative complexity at least $d-1$. This we call the *degree lower bound.*

We say that a circuit is optimal for f if it has $c_\wedge(f)$ AND gates. Since $y \wedge (x \oplus 1) = (y \wedge x) \oplus y$, optimal circuits need not have more than one negation. If present, we may assume this negation is the last gate in the circuit. It is not hard to see that optimal circuits for a Boolean function $f(\mathbf{x})$ require a negation if and only if $f(\mathbf{0}) = 1$, which holds if and only if the polynomial of f has a constant term. Thus we may divide Boolean functions into "positive" functions (those for which $f(\mathbf{0}) = 0$) and "negative" functions. There is a bijection $\sigma(f) = f \oplus 1$ between positive and negative functions. Since the bijection preserves multiplicative complexity, we may restrict our study of multiplicative complexity to functions over the basis $(\oplus, \wedge)$. For technical reasons, and without affecting the multiplicative complexity of functions, we allow $\oplus$ gates to contain any number of inputs (at least one). AND gates, though, are restricted to fan-in exactly 2. We call a gate "internal" if its output is not the output to the circuit. We say a circuit is in *Layered Normal Form* (LNF) if i) all inputs go only to $\oplus$ gates; and ii) outputs of all internal $\oplus$ gates are inputs only to $\wedge$ gates. It is not hard to see that all positive functions have optimal circuits in Layered Normal Form.

Logical expressions over the basis $(\wedge, \oplus)$ correspond to arithmetic expressions over GF_2. We will use the latter notation for the most part of this paper: $a \oplus b, a \wedge b, \bar{a}$ will be written $a \oplus b, ab, a \oplus 1$, respectively. The kth elementary symmetric function on n variables $x_1, x_2, \ldots, x_n$ is defined by

$$\Sigma_k^n(x_1, x_2, \ldots, x_n) = \bigoplus_{S \subseteq \{1,\ldots,n\}, |S|=k} \prod_{i \in S} x_i \qquad (1 \le k \le n).$$

For readability we will also use the alternative notations $\Sigma_k^n(\mathbf{x})$ or simply Σ_k^n. It will prove convenient as well to define $\Sigma_0^n = 1$.

A classical result states that every symmetric function can be represented as a sum of elementary symmetric functions (see [16]). Consider, for example, the MAJORITY function on three variables (i.e. the threshold function $T_2^3 = \Sigma_2^3$). $\Sigma_2^3(x_1, x_2, x_3) = x_1 x_2 \oplus x_1 x_3 \oplus x_2 x_3 = (x_1 \oplus x_2)(x_1 \oplus x_3) \oplus x_1$. The last equality establishes $c_\wedge(T_2^3) = 1$, and also serves to show that the algebraic manipulations necessary to obtain optimal circuits may not be obvious.

The following lemmas appear in [4]:

Lemma 1. *Represent the positive integer k as a sum of powers of 2: $k = 2^{i_0} + 2^{i_1} + \ldots + 2^{i_j}$. Each i is a position of a non-zero bit in the binary representation of k. Then for any $n \ge k$, $\Sigma_k^n = \Sigma_{2^{i_0}}^n \Sigma_{2^{i_1}}^n \ldots \Sigma_{2^{i_j}}^n$.*

Lemma 2. *Let $\mathbf{y} = y_k y_{k-1} \ldots y_0$ be the Hamming weight, in binary representation, of the n-bit string $\mathbf{x}$. Then $y_i = \Sigma_{2^i}^n(\mathbf{x})$ for $i = 0, \ldots, k$.*[1]

These show, for example, that $\Sigma_{11}^n = \Sigma_8^n \Sigma_2^n \Sigma_1^n$ for $n \ge 11$, and the Hamming weight of a 10-bit string $\mathbf{x}$ is a string of length 4 whose bits are $\Sigma_8^{10}(\mathbf{x})$, $\Sigma_4^{10}(\mathbf{x})$, $\Sigma_2^{10}(\mathbf{x})$, and $\Sigma_1^{10}(\mathbf{x})$. Finally, we observe that if $g : GF_2^k \to GF_2$ is derived from $f : GF_2^n \to GF_2$ by fixing the values of $n - k$ variables of f, then $c_\wedge(g) \le c_\wedge(f)$. We call g a *restriction* of f.

3 A Tight Lower Bound on the Multiplicative Complexity of Symmetric Functions

Given a Boolean function f over GF_2^n and a subset S of $\{x_1, \ldots, x_n\}$, we denote by $f_{\bar{S}}$ the function obtained from f by complementing the inputs in S. If $f_{\bar{S}} = f$, we say S is *complementable*. We say S is "proper" if $0 < |S| < n$.

Lemma 3. *If a Boolean function f over GF_2^n has multiplicative complexity less than $\lfloor \frac{n-1}{2} \rfloor$, then it has a proper complementable set.*

[1] See also [13].

Proof. Consider an optimal LNF circuit for f. If the circuit has at most $\lfloor \frac{n-1}{2} \rfloor - 1$ AND gates, the number of $\oplus$ gates is at most $k = 2(\lfloor \frac{n-1}{2} \rfloor - 1) + 1 \leq n - 2$ (recall that a circuit in LNF form may have at most one $\oplus$ gate which is not the input to an $\wedge$ gate). Label these gates $\gamma_1, \ldots, \gamma_k$. Define an $n \times k$ matrix $A = (a_{ij})$ over GF_2 as follows: $a_{ij} = 1$ iff x_i is an input to γ_j. Rows of the matrix correspond to inputs of the circuit. Columns correspond to $\oplus$ gates. Since $rank(A) \leq k \leq n - 2$, there is a subset S (with $0 < |S| \leq n - 1$) of the rows whose sum over GF_2^k is $\mathbf{0}$. Since in a LNF circuit all inputs go only to $\oplus$ gates, and each $\oplus$ gate has an even number of inputs from S, S is a complementable set of inputs. □

For a symmetric function f, if a proper set S of cardinality k is complementable, then *every* set of cardinality k is complementable, including the sets $\{x_1, \ldots, x_k\}$ and $\{x_2, \ldots, x_{k+1}\}$. Hence, $\{x_1, x_{k+1}\}$ is also complementable, so any two inputs are complementable. Thus if the Hamming weights of $\mathbf{x}$ and $\mathbf{y}$ have the same parity, then $f(\mathbf{x}) = f(\mathbf{y})$, so f is linear. We have shown

Lemma 4. *If a symmetric Boolean function f has a proper complementable set S, then f must be linear (i.e. $c_\wedge(f) = 0$).*

A lower bound of $\lfloor \frac{n-1}{2} \rfloor$ for non-linear symmetric functions immediately follows. In the full paper, we prove the slightly stronger result:

Theorem 1. *The multiplicative complexity of an $n-$variate non-linear symmetric function is at least $\lfloor \frac{n}{2} \rfloor$.*

4 Hyperplane Restrictions Yield Fractal Lower Bounds

We now describe a new technique which uses the degree lower bound, but often achieves stronger lower bounds. A plane E in GF_2^n can be specified by an equation $\bigoplus_{i \in I_E} x_i = 0$, where $I_E \subseteq \{1, \ldots, n\}$. For notational simplicity, if the index set is empty, we define $\bigoplus_{i \in \phi} x_i = 0$. Given a Boolean function f on n-bits, we denote the restriction of f to the plane E by $f_{\downarrow E}$. Letting $t = Max(I_E)$, we view $f_{\downarrow E}$ as a function on $n - 1$ variables obtained by substituting $\bigoplus_{i \in I_E - \{t\}} x_i$ for x_t in the polynomial for f. There are many ways to obtain a circuit for $f_{\downarrow E}$ from a circuit for f. For C in Layered Normal Form, $C_{\downarrow E}$ will denote the circuit constructed by replacing x_t by all of the other variables in I_E, removing pairs of identical inputs to XOR gates, and repeatedly removing XOR gates with no inputs and unnecessary AND gates. $C_{\downarrow E}$ will be in Layered Normal Form. We now proceed to prove lower bounds by choosing planes which will decrease the number of AND gates in a circuit without decreasing the degree of the function which is computed. The degree lower bound is then applied to the function resulting from the restriction.

Lemma 5. *Suppose f is an $n-$variate function of degree $k > 1$. If $c_\wedge(f) = k - 1 + e$, where $e \geq 0$, then there exist $u \leq e + 1$ planes $E_1, E_2, ..., E_u$ such that the degree of $(\ldots((f_{\downarrow E_1})_{\downarrow E_2})\ldots)_{\downarrow E_u}$ is at most $k - 1$.*

Corollary 1. *Suppose f is an $n-$variate symmetric function of degree $k > 1$. If $c_\wedge(f) = k-1$, then $deg(f_{\downarrow E}) \leq k-1$ for at least two distinct planes E_1, E_2 where E_1 can be specified by $x_n = \bigoplus_{i=1}^{t_1} x_i \quad (t_1 < n)$, and E_2 can be specified using an equation with at most $n-2$ terms in the sum.*

The technique of hyperplane restrictions yields lower bounds on multiplicative complexity which are better than the degree lower bound for many symmetric functions, including all with degree less than $n-1$. We next state some of these bounds. In section 6, the bound given by the following theorem is shown to be tight for Σ^n_{n-2} and Σ^n_{n-3}.

Theorem 2. *Let f be a $n-$variate symmetric function of degree m, with $1 < m < n-1$. Then $c_\wedge(f) \geq m$.*

The proof of Theorem 2 involves one hyperplane restriction. Lemma 5 can be used to prove tighter bounds using successive hyperplane restrictions under certain combinatorial constraints.

Theorem 3. *Let f be a $n-$variate symmetric function of degree m. Suppose $1 < m \leq n-2$ and $n > 4$. Then, if $\binom{n-4}{m-2}$ is even, $\binom{n-3}{m-1}$ is even, and $\binom{n-2}{m}$ is odd, then $c_\wedge(f) \geq m+1$.*

Theorem 4. *Let f be a $n-$variate symmetric function of degree m. If $\binom{n-6}{m-3}$, $\binom{n-5}{m-2}$, and $\binom{n-4}{m-1}$ are even, while $\binom{n-3}{m}$ is odd, then $c_\wedge(f) \geq m+2$.*

Theorem 3 gives the nontrivial lower bound $c_\wedge(\Sigma^8_4) \geq 5$. The set of points in the plane that satisfy the conditions of either Theorem 3 or Theorem 4 form fractals. Figure 1 plots these points for Theorem 3. The hyperplane restriction technique is a general

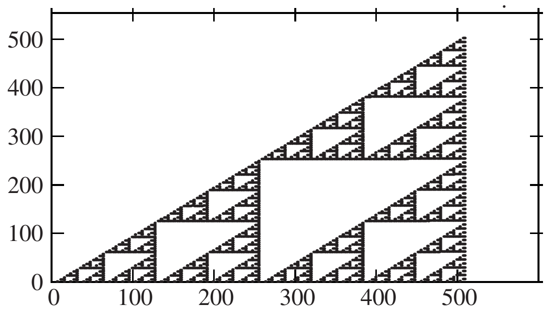


Fig. 1. Points (n,m) for which $c_\wedge(\Sigma^n_m) \geq m+1$, $m < n < 512$

tool for relating combinatorial constraints to multiplicative complexity. The combinatorial constraints thus derived seem to always yield fractals. An interesting question is whether this is solely a result of the bounding technique or the exact complexity of the elementary symmetric functions is in fact fractal in nature.

5 The Exact Multiplicative Complexity of the Hamming Weight Function

The result of computing a symmetric function on some inputs is determined completely by the Hamming weight of those inputs. In this section, we investigate the multiplicative complexity of computing the Hamming weight. Let $\overrightarrow{H}(\mathbf{x})$ denote the binary representation of the Hamming weight of a bit string $\mathbf{x} \in GF_2^n$. $\overrightarrow{H}(\mathbf{x})$ has fixed length $\lceil \log_2(n+1) \rceil$ and may contain leading zeros. The function $\overrightarrow{H}()$ will be denoted by H^n when the parameter n needs to be explicitly stated. Let $H^{\mathbb{N}}(n)$ denote the Hamming weight of the binary representation of the integer n. Theorem 8 in [4] can be seen to give the result that $c_\wedge(H^n) \leq n - H^{\mathbb{N}}(n)$. Here we prove a matching lower bound. It will prove useful to define the Hamming weight of the empty string λ to be 0, i.e. $\overrightarrow{H}(\lambda) = H^{\mathbb{N}}(0) = 0$.

Theorem 5. *$c_\wedge(H^n) = n - H^{\mathbb{N}}(n)$, for all $n \geq 1$.*

Proof. We begin supposing that $\mathbf{x}$ is a bit string of length 2^k. By Lemma 2, the $k+1$st bit of $\overrightarrow{H}(\mathbf{x})$ is $\Sigma_{2^k}^{2^k}(\mathbf{x})$, which is a polynomial of degree 2^k. Thus, by the degree lower bound, $c_\wedge(H^{2^k}) \geq 2^k - H^{\mathbb{N}}(2^k) = 2^k - 1$ for all $k \geq 0$. This matches the upper bound, and these known bounds will now be used to prove the lower bound for lengths which are not powers of 2. For notational brevity, we will denote $c_\wedge(H^n)$ by h_n. Our proof is by induction on k with base $k = 1$. Let $k > 1$ and assume the theorem holds for all $n' \leq 2^{k-1}$. Let $n = 2^k - i$ for some integer $1 \leq i < 2^{k-1}$. Then $n + (i-1) = 2^k - 1$. Note that if $0 \leq a, b, k$ and $n = 2^k - 1 = a + b$, then $H^{\mathbb{N}}(n) = H^{\mathbb{N}}(a) + H^{\mathbb{N}}(b)$. Thus, $k - H^{\mathbb{N}}(i-1) = H^{\mathbb{N}}(n)$. We design a circuit for the Hamming weight of a string $\mathbf{x}$ of length $2^k = n + (i-1) + 1$ as follows. We split $\mathbf{x}$ into three strings $\mathbf{u}, \mathbf{v}, c$ of lengths $n, i-1$, and 1, respectively. We use optimal circuits to compute $\overrightarrow{H}(\mathbf{u})$ and $\overrightarrow{H}(\mathbf{v})$. Note that the longest of these two strings is $\overrightarrow{H}(\mathbf{u})$, which has length k. Then we use the standard addition circuit with carry-in c to compute $c + \overrightarrow{H}(\mathbf{u}) + \overrightarrow{H}(\mathbf{v})$ (which uses k multiplications since a full adder uses just one multiplication for T_2^3). The result is $\overrightarrow{H}(\mathbf{x})$. By the inductive hypothesis, the circuit for $\overrightarrow{H}(\mathbf{v})$ contains $h_{i-1} = (i-1) - H^{\mathbb{N}}(i-1)$ multiplications. Thus the circuit for $\overrightarrow{H}(\mathbf{x})$ contains $h_n + (i-1) - H^{\mathbb{N}}(i-1) + k$ multiplications. Since $c_\wedge(H^{2^k}) \geq 2^k - 1$, this quantity must be at least $2^k - 1$, i.e.

$$h_n + (i-1) - H^{\mathbb{N}}(i-1) + k \geq 2^k - 1.$$

Substituting $H^{\mathbb{N}}(n)$ for $k - H^{\mathbb{N}}(i-1)$, n for $2^k - i$, and rearranging terms, we obtain $h_n \geq n - H^{\mathbb{N}}(n)$. This proves the theorem since the lower bound matches the upper bound from [4]. □

Truncated Hamming weight. Let H_r^n be the function which computes the r low-order bits of the Hamming weight of a vector of length $n \geq 2^{r-1}$. The complexity of this function is 0 when $r = 1$ and $n - H^{\mathbb{N}}(n)$ when $n \leq 2^r - 1$. A recursive construction (see the full paper) yields the following results:

Lemma 6. *For $j \geq r \geq 1$, we have $c_\wedge(H_r^{2^j-1}) \leq \left(\frac{2^{r-1}-1}{2^{r-1}}\right) 2^j - r + 1$.*

Lemma 7. *Let* $r \geq 1$ *and* $n \geq 2^r$. *Let* $\gamma = n \bmod 2^r$. *Then,* $c_\wedge(H_r^n) \leq \left(\frac{2^{r-1}-1}{2^{r-1}}\right)(n-\gamma) + \gamma - H^{\mathbb{N}}(\gamma)$.

6 Building Blocks

We now discuss subclasses of symmetric functions. The idea is to bound, as tightly as possible, the multiplicative complexity of classes of functions which can be used to construct arbitrary symmetric functions. We focus on three classes of functions:

- The elementary symmetric functions $\Sigma_k^n(\mathbf{x})$.
- The "counting" function $E_k^n(\mathbf{x})$, which is 1 if and only if the Hamming weight of $\mathbf{x}$ is k.
- The "threshold" function $T_k^n(\mathbf{x})$, which is 1 if and only if the Hamming weight of $\mathbf{x}$ is k or more.

First, we consider the elementary symmetric functions, Σ_k^n. Let $c_\wedge(f_1, \ldots, f_k)$ denote the multiplicative complexity of simultaneously computing $f_1, \ldots, f_k$. An immediate corollary of Lemma 7 is the following:

Corollary 2. *Let* $r \geq 1$, $n \geq 2^{r-1}$, *and* $\gamma = (n \bmod 2^r)$. *Then*

$$c_\wedge(\Sigma_{2^0}^n, \ldots, \Sigma_{2^{r-1}}^n) \leq \left(\frac{2^{r-1}-1}{2^{r-1}}\right)(n-\gamma) + \gamma - H^{\mathbb{N}}(\gamma).$$

By Lemma 1, the value of $\Sigma_k^n(\mathbf{x})$ is simply the GF_2 product of at most $H^{\mathbb{N}}(k)$ of the low-order $\lceil \log_2(k+1) \rceil$ bits of the Hamming weight of $\mathbf{x}$. Therefore, Corollary 2 yields a general upper bound for Σ_k^n and a less general result:

Theorem 6. *Let* $n \geq k \geq 1$, *and* $r = \lceil \log_2(k+1) \rceil$. *Let* $\gamma = (n \bmod 2^r)$. $c_\wedge(\Sigma_k^n) \leq \left(\frac{2^{r-1}-1}{2^{r-1}}\right)(n-\gamma) + \gamma - H^{\mathbb{N}}(\gamma) + H^{\mathbb{N}}(k) - 1$.

Corollary 3. *For* $n \geq 4$ *and* $n' = n \bmod 4$,
$c_\wedge(\Sigma_4^n) \leq c_\wedge(\Sigma_2^n, \Sigma_4^n) \leq \frac{3}{4}n' + \lfloor \frac{n \bmod 4}{2} \rfloor$.

For example, Corollary 3 yields the result $c_\wedge(\Sigma_4^5) = 3$, though this upper bound also follows from Theorem 5, since $\Sigma_4^5(\mathbf{x})$ is the high-order bit of $\overrightarrow{H}(\mathbf{x})$. We now state several results for the complexity of Σ_k^n for various specific values of k.

Theorem 7. $c_\wedge(\Sigma_2^n) = \lfloor \frac{n}{2} \rfloor$ *and* $c_\wedge(\Sigma_3^n) = \lceil \frac{n}{2} \rceil$.

Lemma 8. *If* m *is odd and* $1 \leq m \leq n$, *then* $\Sigma_m^n = \Sigma_{m-1}^{n-1}\Sigma_1^n$ *and therefore* $c_\wedge(\Sigma_m^n) \leq c_\wedge(\Sigma_{m-1}^{n-1}) + 1$.

Lemma 9. $c_\wedge(\Sigma_{n-1}^n) = n-2$, $c_\wedge(\Sigma_{n-2}^n) = n-2$ *for* $n > 3$, *and* $c_\wedge(\Sigma_{n-3}^n) = n-3$ *for* $n > 4$.

We now turn to the counting and threshold functions, $E_k^n(\mathbf{x})$ and $T_k^n(\mathbf{x})$. The degree of $E_k^n = a_0\Sigma_0^n \oplus \ldots \oplus a_n\Sigma_n^n$ is the largest i such that a_i is non-zero. It is clear that $a_i = 0$ for $i < k$. It turns out there is a simple formula for the remaining a_i.

Lemma 10. $E_k^n = \bigoplus_{i=k}^n a_i \Sigma_i^n$, *where* $a_i = \binom{i}{k} \bmod 2$.

Thus, the expansions of the exactly-k functions can be "read off" rows of Sierpinsky's gasket. For example the expansion of E_6^{13} corresponds to the sixth column (1 1 0 0 0 0 0 0) of the fractal: $E_6^{13} = \Sigma_6^{13} \oplus \Sigma_7^{13}$. Now, $\Sigma_6^{13} \oplus \Sigma_7^{13} = \Sigma_4^{13} \cdot \Sigma_2^{13} \cdot (1 \oplus \Sigma_1^{13})$. Thus $c_\wedge(E_6^{13}) \leq c_\wedge(\Sigma_4^{13}, \Sigma_2^{13}) + 2$. By Corollary 3, $c_\wedge(\Sigma_4^{13}, \Sigma_2^{13}) \leq 9$. Therefore $c_\wedge(E_6^{13}) \leq 11$. This is quite remarkable given the general upper bound of $13 + 3\sqrt{13} > 23$ from [4] (or if one considers that the associated polynomial has over 18 thousand multiplications).

A similar lemma holds for the threshold functions since T_k^n can be expressed recursively using $T_k^n = x_n E_{k-1}^{n-1} \oplus T_k^{n-1}$, which says that at least k of $x_1, \ldots, x_n$ are ones if and only if at least k out of $x_1, \ldots, x_{n-1}$ are ones or (exclusive) x_n is one and exactly $k-1$ out of $x_1, \ldots, x_{n-1}$ are ones. This leads to the following characterization of the expansion of T_k^n based on Sierpinski's gasket.

Lemma 11. $T_k^n = \bigoplus_{i=k}^n b_i \Sigma_i^n$ *where* $b_i = \binom{i-1}{k-1} \pmod 2$.

Since $E_k^n(\mathbf{x}) = E_{n-k}^n(\bar{\mathbf{x}})$, we have $c_\wedge(E_k^n) = c_\wedge(E_{n-k}^n)$ for $0 \leq k \leq n$. Then the degree lower bound yields $c_\wedge(E_k^n) \geq \max\{k-1, n-k-1\}$. Similarly, since $T_k^n(\mathbf{x}) = 1 \oplus T_{n-k+1}^n(\bar{\mathbf{x}})$, we have $c_\wedge(T_k^n) = c_\wedge(T_{n-k+1}^n)$ for $1 \leq k \leq n$, and the degree lower bound yields $c_\wedge(T_k^n) \geq \max\{k-1, n-k\}$. Since $T_n^n = \Sigma_n^n$, we have $c_\wedge(T_1^n) = c_\wedge(T_n^n) = c_\wedge(\Sigma_n^n) = n-1$.

As mentioned above, the degree of E_k^n (or T_k^n) will be the largest value j such that the expansion of E_k^n (T_k^n) contains the term Σ_j^n. In the case of E_k^n this will be the largest $k \leq j \leq n$ such that the binomial coefficient $a_j = \binom{j}{k}$ is odd, and in the case of T_k^n this will be the largest $k \leq j \leq n$ such that $b_j = \binom{j-1}{k-1}$ is odd. Thus, the degree of T_k^n is one more than the degree of E_{k-1}^{n-1}. Given this relation, we will only consider the degree of E_k^n.

A theorem by Kummer [8] shows that the binomial coefficient $\binom{j}{k}$ is odd if and only if $k \sqsubseteq j$, where the notation $k \sqsubseteq j$ means that if the binary representations of k and j are $k_s k_{s-1} \ldots k_1$ and $j_s j_{s-1} \ldots j_1$, respectively, then for each i such that $k_i = 1$, it also the case that $j_i = 1$. This can be used to give the following degree lower bounds on the multiplicative complexity of the exactly-k and threshold-k functions:

Theorem 8. $c_\wedge(E_k^n) \geq \max\{k-1, n-k-1, 2^{\lfloor \log_2 n \rfloor} - 2, l_{n,k} - 1\}$ *and* $c_\wedge(T_k^n) \geq \max\{k-1, n-k, 2^{\lfloor \log_2 n \rfloor} - 1, l_{n-1,k-1}\}$, *where* $l_{n,k}$ *is the bitwise OR of* $n-k$ *and* k.

We now turn to upper bounds. We develop new techniques for producing circuits with few AND gates. We refer to a set of Boolean functions on n variables as a *complete basis* if any symmetric function can be expressed as a linear combination of these functions. Examples of complete bases are $\{\Sigma_i^n \mid 0 \leq i \leq n\}$, and $\{E_i^n \mid 0 \leq i \leq n\}$. Define $A_m^q = \bigoplus_{i=m}^q \Sigma_i^n$ for $m \leq q \leq n$.[2] Then $\Sigma_n^n = A_n^n$ and $\Sigma_m^n = A_m^n \oplus A_{m+1}^n$ for $m < n$. Therefore, $\{A_i^n \mid 0 \leq i \leq n\}$ is complete basis. We will prove upper bounds on the multiplicative complexity of several classes of functions by constructing circuits for functions in the class A_i^q with $0 \leq i \leq q \leq n$.

[2] Note that, in the notation A_m^q, the parameter n is implicit.

Lemma 12. *Let $r \geq 1$ and $2^r - 1 \leq n$. Assume the values of $\Sigma^n_{2^i}$ are known for $i = 0, \ldots, r-1$. Then $A_0^{2^r-1}$ can be computed using $r-1$ additional AND gates.*

Corollary 4. *Let $r \geq 1$ and $2^r - 1 \leq n$. Assume the values of $\Sigma^n_{2^i}$ are known for $i = 0, \ldots, r-1$. Then the functions $A_0^{2^s-1}$ $(0 \leq s \leq r)$ can be simultaneously computed using at most $r-1$ additional AND gates.*

We view the set of functions $\{A_0^{2^s-1} \mid 0 \leq s \leq r\} \cup \{\Sigma^n_{2^i} \mid i = 0, \ldots, r\}$ as a basis. The number of AND gates sufficient to compute any linear combination of functions in this basis is no more than $c_\wedge(H^n_r) + r - 1$.[3] The following corollary allows us to expand the basis.

Corollary 5. *Let $r \geq 0$ and $2^r - 1 \leq n$. Assume the values of $\Sigma^n_{2^i}$ are known for $i = 0, \ldots, r-1$. Then the basis $\{A_0^{2^s-1} \mid 0 \leq s \leq r\} \cup \{A_m^{2^s-1} \mid 0 \leq s \leq r, m = 2^q, q < s\} \cup \{A_m^{2^s-1} \mid 0 \leq s \leq r, m = 2^q + 1, q < s\}$ can be computed using $r-1$ additional AND gates.*

Examples of results obtained using this basis are:

Lemma 13. *Any symmetric function on 7 inputs has multiplicative complexity at most 8.*

Corollary 6. *Let $r \geq 1$, $n = 2^r - 1$, and $m = 2^{r-1}$. Then $c_\wedge(E^n_m) = n - 1$.*

The majority function, a special case of the threshold function, is of particular importance in applications of this theory (e.g. electronic voting protocols). The first two results below give bounds for the majority function and the third general result on threshold functions is obtained using similar techniques.

Theorem 9. *Let $n = 2^r$ and $m = 2^{r-1} + 1$. Then $c_\wedge(T^n_m) = n - 1$.*

Theorem 10. $c_\wedge(T^{2m-1}_m) \leq 2^{\lceil \log_2 m \rceil} + m - \lceil \log_2 m \rceil - 2$ *for all* $m \geq 2$.

Theorem 11. $c_\wedge(T^n_m) \leq n - H^{\mathbb{N}}(n) + \lceil \log_2(n+1) \rceil - 1$ *for all* $m \geq 1$.

References

1. A. A. Aleksanyan. On realization of quadratic Boolean functions by systems of linear equations. *Cybernetics*, 25(1):9–17, 1989.
2. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pages 1–10, 1988.
3. J. Boyar, I. Damgård, and R. Peralta. Short non-interactive cryptographic proofs. *Journal of Cryptology*, 13:449–472, 2000.
4. J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science*, 235:43–57, 2000.
5. P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, 1997.

[3] H^n_r is defined in section 5.

6. R. Cramer, I. Damgård, and J. B. Nielsen. In *EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–300. Springer-Verlag, 2001.
7. D. Chaum, C. Crépeau, and I. Damgård. Multi-party unconditionally secure protocols. In *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pages 11–19, 1988.
8. E. E. Kummer. Über die Ergänzungssätze zu den allgemeinen Reciprocitätsgesetzen. *J. Reine Angew. Math.*, 44:93–146, 1852.
9. J.B. Nielsen and M. Hirt. Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation. In *ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 79–99. Springer-Verlag, 2005.
10. M. V. Mihaĭljuk. On the complexity of calculating the elementary symmetric functions over finite fields. *Sov. Math. Dokl.*, 20:170–174, 1979.
11. R. Mirwald and C. Schnorr. The multiplicative complexity of quadratic Boolean forms. *Theoretical Computer Science*, 102(2):307–328, 1992.
12. W. J. Paul. A $2.5n$ lower bound on the combinational complexity of boolean functions. In *Proceedings of the 7th ACM Symposium on the Theory of Computing*, pages 27–36, 1975.
13. R. Rueppel and J. Massey. The knapsack as a nonlinear function. In *Abstracts of papers, IEEE Int. Symp. on Information Theory*, page 46, 1985.
14. C. P. Schnorr. The multiplicative complexity of Boolean functions. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 6th International Conference*, volume 357 of *Lecture Notes in Computer Science*, pages 45–58, 1989.
15. L. Stockmeyer. On the combinational complexity of certain symmetric Boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.
16. B. L. van der Waerden. *Algebra*. Frederick Ungar Publishing.

On the Complexity of Limit Sets of Cellular Automata Associated with Probability Measures

Laurent Boyer[1], Victor Poupet[1], and Guillaume Theyssier[2]

[1] LIP (UMR 5668 — CNRS, ENS Lyon, UCB Lyon, INRIA), ENS Lyon, 46 allée d'Italie, 69364 LYON cedex 07 France
laurent.boyer@ens-lyon.fr, victor.poupet@ens-lyon.fr
[2] LAMA (UMR 5127 — CNRS, Université de Savoie), Université de Savoie, Campus Scientifique, 73376 Le Bourget-du-lac cedex France
guillaume.theyssier@univ-savoie.fr

Abstract. We study the notion of limit sets of cellular automata associated with probability measures (μ-limit sets). This notion was introduced by P. Kůrka and A. Maass in [1]. It is a refinement of the classical notion of ω-limit sets dealing with the typical long term behavior of cellular automata. It focuses on the words whose probability of appearance does not tend to 0 as time tends to infinity (the persistent words). In this paper, we give a characterization of the persistent language for non sensitive cellular automata associated with Bernoulli measures. We also study the computational complexity of these languages. We show that the persistent language can be non-recursive. But our main result is that the set of quasi-nilpotent cellular automata (those with a single configuration in their μ-limit set) is neither recursively enumerable nor co-recursively enumerable.

1 Introduction

Cellular automata (CA for short) are discrete dynamical systems given by a very simple syntactical definition. They consist of a large collection of identical cells which evolve according to uniform local interactions. Despite the simplicity of the model, they are capable of producing a wide range of different behaviors. One of the main challenges in the field is to give pertinent classifications of these dynamical systems.

There has been a huge amount of attempts in the literature (see [2,3,4]). Among them, the notion of ω-limit set has received a great interest since the results obtained by K. Čulik *et al.* in [5]. This notion (which comes from classical dynamical systems theory) is an attempt to catch the long term behavior of cellular automata. More precisely, the ω-limit set is the set of configurations that may appear in the evolution after an arbitrarily long time. From a topological point of view, it is also the largest attractor. As shown by J. Kari, ω-limit sets can hold a great complexity since any non-trivial property concerning them is undecidable [6]. Among such properties, the nilpotency is the simplest one: a CA is nilpotent if its ω-limit set is reduced to a single configuration. This property is extremely strong since it implies that all initial configurations lead to the same uniform configuration.

The major drawback of ω-limit sets is that they give the same importance to all configurations. Thus, a negligible set of configurations can influence the ω-limit set of a CA and hide properties of its "typical" behavior.

Recently, P.Kůrka and A. Maass introduced in [1] a notion of limit set associated with a probability measure (μ-limit set). Intuitively, this notion catches the "typical" long term behavior of CA. More precisely, it is defined from the patterns whose probability of appearance doesn't go to 0 as time goes to infinity. So, as opposed to classical limit sets, it does not deal with what *may* appear in the long term behavior but focuses on what *does* typically appear. This difference makes the μ-limit set more suitable to study some dynamics (see [1]). Moreover, it is a better tool to give theoretical justifications to many phenomena observed experimentally (since experimentations are not exhaustive, they must restrain to "typical" orbits).

In this paper, we mainly study this set from a computational complexity point of view. We first give a new characterization of μ-limit sets associated with Bernoulli measures for any non sensitive CA. Our characterization shows that the μ-limit set does not depend on the measure.

Then we focus on the quasi-nilpotency property: a CA is μ-quasi-nilpotent if its μ-limit set is reduced to a single configuration. One can think that the undecidability behind limit sets disappears as soon as we no longer consider all configuration but only "typical" ones. We show that this is not the case, the Turing degree of the quasi-nilpotency problem is even higher than that of the nilpotency problem: the set of quasi-nilpotent CA is neither recursively enumerable nor co-recursively enumerable. The construction used to obtain this result also allows us to show that some CA have a non recursive μ-limit language.

2 Definitions

Formally, a *one-dimensional CA* $\mathcal{A}$ is a triple $(Q_\mathcal{A}, r, \delta_\mathcal{A})$, where $Q_\mathcal{A}$ is a finite set of states called the *alphabet*, r is the *radius* and $\delta_\mathcal{A} : Q_\mathcal{A}^{2r+1} \to Q_\mathcal{A}$ is the *local rule*. A *configuration* c describes the state of all cells at a given time: this is a mapping from $\mathbb{Z}$ to $Q_\mathcal{A}$. The set of all possible configurations is denoted $Q_\mathcal{A}^\mathbb{Z}$. For $c \in Q_\mathcal{A}^\mathbb{Z}$, we will often denote by c_z the value of c at $z \in \mathbb{Z}$.

The local description of the CA induces a global evolution. At every step of the computation, the configuration changes according to the global transition rule $G_\mathcal{A} : Q_\mathcal{A}^\mathbb{Z} \to Q_\mathcal{A}^\mathbb{Z}$ induced by the locale rule as follows:

$$G_\mathcal{A}(c)_i = \delta_\mathcal{A}(c_{i-r}...c_i...c_{i+r}).$$

In the following, when considering a CA $\mathcal{A}$, we implicitly refer to the triple $(Q_\mathcal{A}, r, \delta_\mathcal{A})$, where the same symbol $\mathcal{A}$ denotes both the local and the global mapping.

We denote by $Q_{\mathcal{A}}^* = \bigcup_{n\in\mathbb{N}} Q_{\mathcal{A}}^n$ the set of all finite words over $Q_{\mathcal{A}}$. The *length* of $u = u_1u_2...u_n$ is $|u| = n$, and, $\forall a \in Q_{\mathcal{A}}$, $|u|_a$ is the number of occurences of a in u. $\forall 0 < i \leq j \leq |u|$, we also define $u_{[i,j]} = u_iu_{i+1}...u_j$ and $c_{[i,j]}$ for $c \in Q_{\mathcal{A}}^{\mathbb{Z}}$ in a similar way. A word u is a *factor* of a word v if there exist i and j such that $u = v_{[i,j]}$.

For every $c \in Q_{\mathcal{A}}^{\mathbb{Z}}$, the *language of* c, denoted by $L(c)$, is defined by

$$L(c) = \{u \in Q_{\mathcal{A}}^* : \exists i \in \mathbb{Z}, u = c_{[i,i+|u|-1]}\}.$$

The language of a subset of $Q_{\mathcal{A}}^{\mathbb{Z}}$ is the union of the languages of its elements.

The *limit set* of a CA $\mathcal{A}$ is given by $\Omega_{\mathcal{A}} = \bigcap_{n\in\mathbb{N}} \mathcal{A}^n(Q_{\mathcal{A}}^{\mathbb{Z}})$. Intuitively, a configuration is in the limit set if and only if it may appear after an arbitrarily long evolution. A CA is said to be *nilpotent* if its limit set is reduced to a single configuration.

For every $u \in Q_{\mathcal{A}}$ and $i \in \mathbb{Z}$ we define the *cylinder* $[u]_i$ as the set of configurations containing the word u in position i:

$$[u]_i = \{c \in Q_{\mathcal{A}}^{\mathbb{Z}} : c_{[i,i+|u|-1]} = u\}.$$

Let $\mathcal{A}$ be any CA and μ any Borel probability measure on $Q_{\mathcal{A}}^{\mathbb{Z}}$ (a measure on the Borel sets, *i.e.* the smallet σ-algebra containing open sets). For any $n \geq 0$, $\mathcal{A}^n\mu$ denotes the probability measure such that for any Borel set $U \subseteq Q_{\mathcal{A}}^{\mathbb{Z}}$ we have $\mathcal{A}^n\mu(U) = \mu\big(\mathcal{A}^{-n}(U)\big)$. If $Q_{\mathcal{A}} = \{a_1, \ldots, a_n\}$ is the working alphabet, a Bernoulli measure μ over $Q_{\mathcal{A}}^{\mathbb{Z}}$ is given by a probability vector $(p_1, \ldots, p_n)$ $(0 \leq p_i \leq 1$ and $\sum p_i = 1)$ such that, for any word $u \in Q_{\mathcal{A}}^*$ and any $i \in \mathbb{Z}$, $\mu([u]_i) = \prod_{a\in Q_{\mathcal{A}}} p_a^{|u|_a}$. A Bernouilli measure is *complete* (or with full support) if $p_i \neq 0$ for all i.

Definition 1 (Persistent set). *Let $\mathcal{A}$ be any CA and μ be a Bernoulli measure on $Q_{\mathcal{A}}^{\mathbb{Z}}$. A word $u \in Q_{\mathcal{A}}^*$ is a* vanishing word *for $\mathcal{A}$ and μ if its probability to appear (in a certain position) after n iterations tends to 0 as n grows to infinity. We define the set $L_{\Upsilon,\mu}(\mathcal{A})$ of* persistent words *for $\mathcal{A}$ and μ as the complement of the set of vanishing words for $\mathcal{A}$ and μ: $u \notin L_{\Upsilon,\mu}(\mathcal{A}) \iff \lim_{n\to\infty} \mathcal{A}^n\mu([u]_0) = 0$. Then the μ-*persistent set *or* μ-limit set *of $\mathcal{A}$ is the subshift $\Upsilon_\mu(\mathcal{A})$ defined by $L_{\Upsilon,\mu}(\mathcal{A})$, precisely $\Upsilon_\mu(\mathcal{A}) = \big\{c \in Q_{\mathcal{A}}^{\mathbb{Z}} : L(c) \subseteq L_{\Upsilon,\mu}(\mathcal{A})\big\}$.*

When considering limit sets, the most studied property is certainly the nilpotency. By analogy, we may define the notion of μ-quasi-nilpotency associated with the μ-limit-set.

Definition 2 (Quasi-nilpotency). *Let $\mathcal{A}$ be a CA and μ be any Bernoulli measure over $Q_{\mathcal{A}}^{\mathbb{Z}}$. $\mathcal{A}$ is said to be* μ-quasi-nilpotent *if $\Upsilon_\mu(\mathcal{A})$ is reduced to a single configuration.*

One can verify that a CA $\mathcal{A}$ is μ-quasi-nilpotent if and only if there is some state $q \in Q_{\mathcal{A}}$ such that $L_{\Upsilon,\mu}(\mathcal{A}) = q^*$.

Definition 3 (Walls). *Let $\mathcal{A}$ be any CA. For any $u \in Q_{\mathcal{A}}^*$, we denote by $[u]_{mid}$ the following set of configurations of $Q_{\mathcal{A}}^{\mathbb{Z}}$:*

$$[u]_{mid} = \begin{cases} [u]_{-\frac{|u|}{2}} & \textit{if } |u| \textit{ is even,} \\ [u]_{-\frac{|u|+1}{2}} & \textit{if } |u| \textit{ is odd.} \end{cases}$$

A wall *for $\mathcal{A}$ is a sequence $\mathcal{W} = \big(w_n\big)_{n\geq 0}$ of non empty words of $Q_{\mathcal{A}}^*$ such that:*

1. *$\forall c \in [w_0]_{mid}, \forall n \geq 1 : \mathcal{A}^n(c) \in [w_n]_{mid}$;*
2. *the sequence $\big(|w_n|\big)_{n\geq 0}$ is non-increasing.*

Notice that a wall $\mathcal{W} = \big(w_n\big)_{n\geq 0}$ is necessarily ultimately periodic since $[w_0]_{\text{mid}}$ contains spatially periodic configurations. The word w_0 is said to be the *foot* of $\mathcal{W}$. A word is a *foot of wall* for $\mathcal{A}$ if it is the foot of some wall for $\mathcal{A}$. Any word in the period of the sequence $\mathcal{W}$ will be called a *brick* of $\mathcal{W}$: formally, w is a brick of $\mathcal{W}$ if there are p, n_0 such that, for all $n \in \mathbb{N}$, $w_{pn+n_0} = w$. A word $w \in Q_{\mathcal{A}}^*$ is a *brick of wall* for $\mathcal{A}$ if it is a brick of some wall of $\mathcal{A}$.

The following well-known property relates the existence of bricks of wall to the property of sensitivity to initial conditions (see [3] for a proof).

Proposition 1. *A CA $\mathcal{A}$ of radius r is sensitive to initial conditions if and only if it has no brick of wall of size r.*

The key property behind that proposition is expressed by the following easy-to-prove lemma.

Lemma 1. *Let $\mathcal{A}$ be any CA of radius r. If w is the foot of a wall of $\mathcal{A}$ having some brick of size at least r, then for any word $u \in Q_{\mathcal{A}}^*$ there exists a wall of $\mathcal{A}$ whose foot is wuw and which has bricks of size at least $|u|$.*

3 Properties of Persistent Sets

It is well-known that the limit set of any CA $\mathcal{A}$ is either reduced to a single configuration or infinite. This fact does not hold with μ-limit sets as shown by the following example. The same example shows a CA whose persistent set does not contain any uniform configuration (the limit set always does).

Example 1. Let $\mathcal{A}$ be the 184 CA in Wolfram's notation. That is, a two states ($Q_{\mathcal{A}} = \{0, 1\}$) one dimensionnal CA of radius 1. Its local rule is given by: $\forall x \in \{0, 1\}, \mathcal{A}(1, 0, x) = \mathcal{A}(x, 1, 1) = 1$ and $\mathcal{A}(0, 0, x) = \mathcal{A}(x, 1, 0) = 0$. It can be seen as a simple model of traffic jam (see [7]).

We first show that for the uniform Bernoulli measure μ_0, the words 11 and 00 are both vanishing. It can be easily checked that $u = u_0u_1...u_{2n+1} \in \mathcal{A}^{-n}(00)$ implies that $u_1u_2...u_{2n+1}$ is a left factor of a well-bracketed string (where 0 "opens" and 1 "closes"). As the proportion of such strings among all words of length n tends to 0 as n grows to infinity, $\lim_{n\to\infty} \mathcal{A}^n\mu_0(00) = 0$ and 00 is not persistent. A similar argument shows that 11 is also vanishing.

Because for all n there is at least one word of length n in $L_{\Upsilon,\mu_0}(\mathcal{A})$, and $L_{\Upsilon,\mu_0}(\mathcal{A})$ is stable by factor, and 00 and 11 are not in $L_{\Upsilon,\mu_0}(\mathcal{A})$, we have $L_{\Upsilon,\mu}(184) = (0+\epsilon)(10)^*(1+\epsilon)$ and $\Upsilon_\mu(184) = \{{}^\omega(01)^\omega, {}^\omega(10)^\omega\}$. □

We will now give a characterization of the persistent language of non sensitive cellular automata. Before stating the theorem, we need a lemma expressing that for infinitely many steps the preimages of a persistent word must contain any given word at some fixed position.

Lemma 2. *Let $\mathcal{A}$ be any CA of radius r and μ be any complete Bernoulli measure over $Q_\mathcal{A}^\mathbb{Z}$. Then, for any $w \in Q_\mathcal{A}^*$ and $u \in L_{\Upsilon,\mu}(\mathcal{A})$ there are positive integers k_1 and k_2 and a strictly increasing sequence of positive integers $\left(n_j\right)_{j\geq 0}$ such that*

$$\forall j \geq 0 : \mathcal{A}^{-n_j}(u) \cap \left(Q_\mathcal{A}^{rn_j-k_1-|w|} \cdot \{w\} \cdot Q_\mathcal{A}^{k_1+k_2+|u|} \cdot \{w\} \cdot Q_\mathcal{A}^{rn_j-k_2-|w|}\right) \neq \emptyset.$$

Proof. Suppose by contradiction that $u \in L_{\Upsilon,\mu}(\mathcal{A})$ does not verify the lemma. Then we have $\forall k \geq 0, \exists n_k \geq 0, \forall n \geq n_k$:

$$\mathcal{A}^{-n}(u) \subseteq Q_\mathcal{A}^{n-k|w|}(Q_\mathcal{A}^{|w|} \setminus \{w\})^k Q_\mathcal{A}^{|u|}(Q_\mathcal{A}^{|w|} \setminus \{w\})^k Q_\mathcal{A}^{n-k|w|}.$$

Then, for any k and any $n \geq n_k$, we have : $\mathcal{A}^n\mu([u]_0) \leq \left(1-\mu([w]_0)\right)^{2k}$. Thus, $\mathcal{A}^n\mu(u) \to 0$ as $n \to \infty$ and $u \notin L_{\Upsilon,\mu}(\mathcal{A})$. □

Theorem 1. *Let $\mathcal{A}$ be a CA which is not sensitive to initial conditions and μ any complete Bernoulli measure. Then $L_{\Upsilon,\mu}(\mathcal{A})$ is exactly the set of bricks of wall for $\mathcal{A}$.*

Proof. First, consider a brick of wall u for $\mathcal{A}$. By definition, there exists a word $w \in Q_\mathcal{A}^*$ and positive integers n_0 and p such that $\forall c \in [w]_{\text{mid}}$ and $\forall n \geq 0$: $\mathcal{A}^{np+n_0}(c) \in [u]_{\text{mid}}$. Thus $\mathcal{A}^{np+n_0}\mu([u]_0) \geq \mu([w]_0)$ which proves $u \in L_{\Upsilon,\mu}(\mathcal{A})$.

Conversely, let $u \in L_{\Upsilon,\mu}(\mathcal{A})$. By proposition 1, if $\mathcal{A}$ is not sensitive to initial conditions, it has a brick of wall of size at least r (where r is the radius of $\mathcal{A}$) associated with some wall $\mathcal{W} = \left(w_n\right)_{n\geq 0}$. Applying lemma 2 to w_0, we know there exist positive integers k_1 and k_2 and a strictly increasing sequence of positive integers $\left(n_j\right)_{j\geq 0}$ such that

$$\forall j \geq 0 : \mathcal{A}^{-n_j}(u) \cap \left(Q_\mathcal{A}^{rn_j-k_1-|w_0|} \cdot \{w_0\} \cdot Q_\mathcal{A}^{k_1+k_2+|u|} \cdot \{w_0\} \cdot Q_\mathcal{A}^{rn_j-k_2-|w_0|}\right) \neq \emptyset.$$

Since $Q_\mathcal{A}^{k_1+k_2+|u|}$ is finite, we can extract from $\left(n_j\right)_{j\geq 0}$ a sub-sequence $\left(n_{j_k}\right)_{k\geq 0}$ such that for some $v \in Q_\mathcal{A}^{k_1+k_2+|u|}$ we have:

$$\forall k \geq 0 : \mathcal{A}^{-n_j}(u) \cap \left(Q_\mathcal{A}^{rn_{j_k}-k_1-|w_0|} \cdot \{w_0\} \cdot v \cdot \{w_0\} \cdot Q_\mathcal{A}^{rn_{j_k}-k_2-|w_0|}\right) \neq \emptyset.$$

By lemma 1, w_0vw_0 is the foot of a wall of $\mathcal{A}$ with a brick of size at least $|v|$. By the above property, we conclude that u is a factor of such a brick of wall. Therefore u is itself a brick of wall of $\mathcal{A}$. □

Notice that theorem 1 implies that, for any CA $\mathcal{A}$ which is not sensitive to initial conditions, the set $\Upsilon_\mu(\mathcal{A})$ is the same for any complete Bernoulli measure.

However, there exists some sensitive CA whose μ-persistent set does depend on the Bernoulli measure μ as pointed out by A. Maass and P. Kůrka in [1]: for instance the "just gliders" CA $\mathcal{A}$ is sensitive to initial conditions and such that, for any Bernouilli measure μ, $\Upsilon_\mu(\mathcal{A})$ is reduced to a single configuration if and only if μ gives the same probability to two peculiar letters of $Q_{\mathcal{A}}$.

4 Undecidability Results

This section addresses different decision problems associated with the persistent language of cellular automata. To simplify the statement of the studied problems, we will only consider the uniform measure. Thus, μ will always denote the uniform measure in this section (the working alphabet will be determined by the context). However, all the results extend to complete Bernouilli measures using lemma 2 and theorem 1 from previous section.

Remark 1. In his proof of undecidability of nilpotency [8], J. Kari actually shows that it is undecidable to determine whether a given CA $\mathcal{A}$ with a spreading state (a state s such that $\delta_{\mathcal{A}}(a_1, \dots, a_n) = s$ whenever $s \in \{a_1, \dots, a_n\}$) is nilpotent. Moreover, it follows from theorem 1 that such a CA is μ-quasi-nilpotent for any Bernoulli measure μ (since the only bricks of wall are the words s^n, $n \in \mathbb{N}$). Thus, it is undecidable to determine whether a μ-quasi-nilpotent CA is nilpotent. □

Theorem 2. *The set of μ-quasi-nilpotent CA is not recursively enumerable.*

Proof. Given a Turing machine M of states Q_M and tape alphabet $\Sigma = \{0, 1, B\}$ working on a semi-infinite tape, we will construct a CA $\mathcal{A}$ of radius 1 that will be quasi-nilpotent if and only if M doesn't halt on the empty input.

The states of $\mathcal{A}$ will be $\{\#\} \cup (S_{\text{simul}} \times S_{\text{signals}})$ where # is an inalterable state, meaning that if a cell is in this state it will never change to any other state, $S_{\text{simul}} = (Q_M \cup \{-\}) \times \Sigma$ is the set of states needed to simulate the behavior of M (a state $(-, \alpha)$ represents a cell of the tape containing the letter α without the head and a state (q, α) represents that the head is on this cell in state q) and $S_{\text{signals}} = \{-, L, F, R, D\}$ is a set of *signals* whose meaning and behavior will be explained later.

The transition rule of the automaton can be described by the following rules:

- As said earlier, # states are inalterable. Since the automaton is of radius 1, they act as delimiters or walls, no information can go across them. A finite set of contiguous non-# cells between two # states will be referred to as a segment. The length of the segment will be the number of cells between the two # states.
- At all times, all cells not in the # state will simulate the behavior of M on their first component. We deal with conflicts (two heads that want to move on a cell for example) in any given way, since we'll see that these have no

impact on what we'll do later (ultimately, we'll only be interested in regular simulations starting on an empty input). If at some point in the computation the head wants to move to a cell in state #, the head is deleted so that the computation cannot end.
- The signal $-$ means that there is in fact no particular signal on the cell.
- If at some point in the computation the final state q_f of M is reached, the cell where this state appears generates a signal F (on its "signal" component).
- The F signal moves towards the left at maximum speed. When it reaches the left border of the segment (#) it turns into an R signal.
- The R signal will move to the right and while doing so it will reset the computation that is held on the first component of the cells it moves through, meaning that it will put the head in its initial state q_0 on the first cell of the segment and put a blank symbol B on every cell of the tape. Since this signal moves at maximum speed, the simulation of M can occur without problems on a clean tape.
- When the R signal meets the right end of the segment it disappears.
- During all this time, the rightmost cell of a segment (any cell that is on the left of a # cell) will generate L signals at every time.
- L signals move to the left at maximum speed. When one of these signals reaches the left border of a segment, it generates a D signal.
- The D signals delete the whole segment by moving to the right while changing all the cells they go through into #. They obviously disappear when they meet a # cell since they can't go any further.

All the signals that we use move at maximum speed (one cell per step) in one of the two available directions. Signals going in opposite directions are not allowed to cross each other, thus, one of the two must disappear. The priority is as follows:

$$L < F < R < D$$

For example, if an R signal is moving to the right (while cleaning the computation) and an L signal is moving to the left, when they meet the R signal keeps moving to the right and the L signal disappears.

Let's assume that M halts in t steps and let's consider the segment of length $2t$ in which there are no signals on any cell, the first cell is in state (q_0, B) and all other cells are in state $(-, B)$. On this segment, the simulation of M starts from a well formed configuration so it will reach the q_f state after t steps and generate an F signal. Meanwhile L signals appear from the right border and move to the left. Because the segment is of length $2t$, the F signal appears on the left of all L signals, so it reaches the origin before all L signals and creates an R signal. This R signal will reset the computation while deleting all L signals. From there a new computation starts that will have enough time to finish again and delete the L signals again. Because the segment is "protected" from any outside interference by the # cells, this cycle will continue forever and no # state will appear on the segment. Because there are only a finite number of possible configurations on the segment the automaton eventually enters a cycle on this non-empty segment.

According to theorem 1 this segment is part of the persistent language so $\mathcal{A}$ is not μ-quasi-nilpotent.

Now we will assume that M doesn't halt and show that any segment of length n disappears after at most $5n$ steps. The proof is based on the observation that we can't delay the apparition of a D signal on the first cell of the segment for more than $4n$ steps.

It's possible that there was already a D signal somewhere on the segment in the inital configuration. In this case, the D signal will cut the segment in two by creating a # state where it was initially and then delete the right part of the segment. This means that if there is a D signal on a segment in the initial configuration we can focus on a shorter segment on which there is no D initially and let the already present D take care of the rest of the segment.

Therefore we can assume that the segment we are studying doesn't contain any D initially. This means that after at most n steps all original R signals will have disappeared. From there, L signals will start appearing on the right border of the segment and try to proceed to the left (they would arrive at time $2n$). To stop them from reaching the left border and generating a D signal, the only possibility is to generate an R signal on the left border of the segment before the time $2n$. From there, the R signal will reset the configuration of the simulation so that what is computed on the left of this R signal is a normal computation of M on the empty input. Since we have assumed that M doesn't halt, this "well formed" computation will not reach the q_f state. When the R signal reaches the right end of the segment (at time at most $3n$), it disappears and the L signals start moving to the left again. Since the simulation of M doesn't reach the final state no F signal is generated so there's nothing to stop the L signals from reaching the left border, generate a D signal and delete the whole segment. The whole segment is therefore deleted after at most $5n$ steps.

To complete the proof, we need only show that in this case (if M doesn't halt) no other state than # can appear in a brick of wall. Let's consider a wall $\mathcal{W} = (w_i)_{i\in\mathbb{N}}$. Let's consider the configuration c_{w_0} containing # states everywhere except on its center where it is the word w_0. Obviously c_{w_0} is in $[w_0]_{\text{mid}}$ and doesn't contain any segment longer than $|w_0|$ so no segment will survive more than $5|w_0|$ steps, which means that for any $n \geq 5|w_0|$, $\mathcal{A}^n(c_{w_0})$ is the uniform # configuration, which implies that $w_n \in \#^*$. From theorem 1 we conclude that $\mathcal{A}$ is μ-quasi-nilpotent. □

Corollary 1. *Given a CA $\mathcal{A}$ and a word w, the property that w is not persistent for $\mathcal{A}$ is not semi-decidable. In other words the set $\{(\mathcal{A}, w) | w \notin L_{\Upsilon,\mu}(\mathcal{A})\}$ is not recursively enumerable.*

Proof. We know that a CA is quasi-nilpotent if and only if only one of its states is persistent. If we could semi-decide that a given state is not persistent, then we could use this algorithm on all states in parallel and if the CA is quasi-nilpotent the algorithm would eventually show that all but one states are not persistent, thus showing that the CA is quasi-nilpotent. We would therefore have an algorithm to semi-decide that a CA is quasi-nilpotent, which is in contradiction with theorem 2. □

Remark 2. The proof above shows that it is also undecidable to determine whether the persistent set is finite or not. Indeed, it is not difficult to check that the persitent set of the constructed CA is either reduced to a single configuration or infinite. □

Theorem 3. *There exists a CA with a non-recursive persistent language.*

Proof (sketch). It is possible to show this by slightly modifying the CA constructed in the proof of theorem 2. To do so we use another layer in the states so that each regular cell of a segment also has a "memory" containing a tape symbol. The memory of a cell can never be changed (except when the cell becomes # in which case the memory is lost). Instead of starting from an empty input when the simulation of M is reset by an R signal it's the memory of each cell that's written on the tape. This way we can simulate the behavior of M on any input. It is then easy to prove that a segment survives if and only if the memory of its cells corresponds to a word wB^k where $M(w)$ ends using less than $|w| + k$ cells.

If the persistent language of $\mathcal{A}$ is recursive, then the language $\#wB$ such that $M(w)$ halts is also recursive: there is a segment in the persistent language whose memory layer is wB^k, so there is a word of memory $\#wB$ (stability by factor). Therefore if the domain of M is not recursive neither is $L_{\Upsilon,\mu_0}(\mathcal{A})$.

Theorem 4. *The set of μ-quasi-nilpotent CA is not co-recursively enumerable.*

Proof. As with the proof of theorem 2, we will consider a Turing machine M and create a cellular automaton $\mathcal{A}$ of radius 1 that simulates M. $\mathcal{A}$ will be quasi-nilpotent if and only if M halts on the empty input. The idea is that we will again simulate the behavior of M on each segment but now if the simulation doesn't halt the right # of the segment will be erased so that the available space for the simulation is increased, and the simulation will start again. If at some point the simulation ends then the segment is erased. This way non-empty segments will remain on the configuration if the machine M doesn't halt but almost every segment will be deleted if the machine halts.

The construction will be very similar to the one of the proof of Theorem 2. The states of $\mathcal{A}$ are now $\{\#\} \cup (S_{\text{simul}} \times S_{\text{signals}} \times \{0, 1\})$, the new set of signals being $S_{\text{signals}} = \{-, L, R, D, D_L, D_R, C_L, C_R\}$. The added bit doesn't affect the computation and never changes on a cell. We'll call it *neutral bit.*

The evolution of the automaton is described as follows:

- The # state is now "almost" inalterable in the sense that only one particular signal (D) can erase it. We will continue to use the notion of segment (finite set of contiguous cells between two #).
- The simulation of M takes place on each segment as in the previous construction.
- L signals will appear continuously on the right border of a segment and proceed to the left.
- When an L signal meets the # cell at the left border of the segment it turns into a D signal.

- D signals move to the right. They erase all L signals they meet. If a D signal finds a final state q_f in the simulation of M, it generates two signals D_L and D_R that will erase the segment (turn all cells into #) by propagating to the left and right respectively until they reach the end of the segment. If the D signal doesn't see any q_f state and reaches the right # of the segment it turns it into a regular cell whose neutral bit is the same as its left neighbor and creates two signals C_L and C_R on the cells next to where the # cell was.
- The C_L and C_R signals move to the left and to the right respectively. Their function is to clear the segment so that a fresh simulation of M can start back from the beginning. Both signals will erase any signal they come across, C_L having the priority over C_R. When the C_L signal reaches the beginning of the segment it turns into an R signal. When the C_R signal reaches the end of the segment it disappears.
- The R signal moves to the right and resets the simulation as it moves as in the previous proof. It also erases all L signals.

Proving the theorem from this construction will now be similar to the proof of theorem 2. The # states can only be deleted by a signal that comes from their left so if two segments merge it's because the merging signal came from the leftmost of the two segments while the rightmost one can do nothing to prevent it. We'll say that the the left segment *invades* the right one.

Let's see what happens if M doesn't halt on empty input. In that case a "normal" simulation of M will never reach the q_f state so the D_L and D_R signals should never appear. On any segment where there are initially no signals and no simulation of M going on L signals will appear, reach the left border, and generate an R signal that will start a new correct simulation. This simulation will not end so the segment will eventually invade the one on its right and when doing so C_L and C_R signals will appear to clean the segment and a new correct simulation will again take place on the wider segment, etc. Since no matter how wide the segment is the simulation will never end the segment will never disappear. It is also possible that the segment we have considered is eventually invaded but when the invasion occurs C_L and C_R signals appear that will clean the wider segment and ensure that the new simulation that takes place on this segment is also correct so again there's no risk that the segment disappears.

In other words, if M doesn't halt, any "inactive" segment on the initial configuration will grow and survive forever (the cells that were initially on this segment will never become #). Let s be such an "inactive" segment of length $2k+1$ including the border #, then for all $n \in \mathbb{N}$ and all $w \in Q_{\mathcal{A}}^n$,

$$wsw \in \bigcup_{q \in Q_{\mathcal{A}} \setminus \{\#\}} A^{-n-k}(q)$$

This means that $\sum_{q \in Q_{\mathcal{A}} \setminus \{\#\}} \mathcal{A}^n \mu(q) \geq \mu([s]_0)$ so at least one of the non-# states is persistent. By symmetry, changing the neutral bit doesn't change the persistent nature of the state so we have at least two persistent states and $\mathcal{A}$ is not quasi-nilpotent.

Now we have to check that if the machine M halts in t steps then no other state than # is persistent. Let's consider a segment s of length $l \geq 2t$ on which there is no simulation of M going on and the only signal present is an R signal on the first cell. While this segment is not invaded, it will simply do correct simulations of M, reach the final state in time so that the D signal sees it, the D_L and D_R signals will therefore appear and turn the whole segment into #. This means that such a segment doesn't invade its right neighbor. Moreover, if it happens to be invaded by its left neighbor, the invasion will make C_L and C_R segments appear, which will ensure that on this new segment a new correct simulation starts. The segment will do correct simulations and grow until it's big enough so that a simulation ends. This will happen before all the # from s's disappearance have been deleted because there will be enough room to complete a simulation before that so this other segment will also turn to # before going past the initial boundaries of s. This means that if M terminates there exists a segment such that no matter what happens it will never invade its right neighbor. We'll call such a segment *non-invasive.*

Let's see what happens to a segment such that there is a non-invasive segment on its left at a distance d_l (the distance is taken from the right border of the non-invasive segment to the left border of the considered segment) and a # on its right at a distance $d_r \geq 2t$ (taken from the right border of the segment).

While the segment is not invaded it will after some time that we can bound easily depending on its length start a correct simulation or be completely deleted (because the L signals cannot be delayed forever). From there it will continuously do simulations and invade its neighbors if the simulations do not halt. Since there is a # at a distance $d_r \geq 2t$, the simulation will eventually end before this # symbol is deleted since the segment will be wide enough, so the segment will eventually disappear. The only thing that could delay the disappearance of the segment would be a series of invasion of the segment. However, since there is a non-invasive segment on the left of segment, we know that there is only a limited number of possible invasions so we can bound the time until all possible invasions have occurred. From there, the simulation will start correctly on a segment that will not be invaded and will therefore disappear.

To sum up, we have shown that if M halts, there exists a function $\sigma : \mathbb{N}^2 \to \mathbb{N}$ such that any segment that has a non-invasive segment on its left at a distance d_l and a # cells on its right at a distance $d_r \geq 2t$, will disappear after at most $\sigma(d_l, d_r)$ steps. This means that for any $n \geq \sigma(d_l, d_r)$ and any $q \in Q_{\mathcal{A}} \setminus \{\#\}$, any word in $\mathcal{A}^{-n}(q)$ has no non-invasive segment on the cells left of the position $-d_l$ and no two # symbols on the cells between positions $2t$ and d_r (the first one is the end of the segment, that can possibly be deleted by an already-present D signal). This restriction implies that none of these states is persistent (see lemma 2). □

Corollary 2. *Given a CA $\mathcal{A}$ and a word w, the property that w is persistent for $\mathcal{A}$ is not semi-decidable. In other words the set $\{(\mathcal{A}, w) | w \in L_{\Upsilon,\mu}(\mathcal{A})\}$ is not recursively enumerable.*

5 Conclusion and Perspectives

We proved that the μ-quasi-nilpotency property is neither recursively enumerable nor co-recursively enumerable. In our opinion, such a result has two interesting aspects. First, it deals with a kind of problem rarely considered in the literature: a property of "typical" or random configurations only. We believe that such properties are closer to what experimental observations may capture and therefore that our undecidability results have a stronger meaning to physicists or other scientists concerned with modelling using cellular automata. Second, it gives an example of a "natural" property of cellular automata with a high Turing degree (few examples are known, see [9]).

A natural way to continue the study of the computational complexity of persistent sets would be to try to prove a Rice theorem for μ-limit sets. Any property concerning limit sets is either trivial or undecidable. Is it the same for μ-limit sets?

Another interesting research direction would be to understand better how the probability of appearance of some word can vary with time. More precisely, we left open a very simple question: do we have $L_{\Upsilon,\mu}(\mathcal{A}) = L_{\Upsilon,\mu}(\mathcal{A}^t)$ for any CA $\mathcal{A}$ and any t ?

Finally, we can also consider extensions of our work to a broader class of measures or by raising the dimension. In the latter case, the notion of wall does not play the same role (a finite pattern does not cut a bi-dimensional configuration into two disconnected components) and the case of non-sensitive CA is to be reconsidered.

References

1. Kůrka, P., Maass, A.: Limit Sets of Cellular Automata Associated to Probability Measures. Journal of Statistical Physics **100** (2000) 1031–1047
2. Wolfram, S.: Universality and complexity in cellular automata. Physica D **10** (1984) 1–35
3. Kůrka, P.: Languages, equicontinuity and attractors in cellular automata. Ergodic Theory and Dynamical Systems **17** (1997) 417–433
4. Mazoyer, J., Rapaport, I.: Inducing an Order on Cellular Automata by a Grouping Operation. In: Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science (1998)
5. Čulik, II, K., Pachl, J., Yu, S.: On the limit sets of cellular automata. SIAM Journal on Computing **18** (1989) 831–842
6. Kari, J.: Rice's theorem for the limit sets of cellular automata. Theoretical Computer Science **127** (1994) 229–254
7. Nagel, K., Schreckenberg, M.: A cellular automaton model for freeway traffic. J. Phys. **2** (1992) 2221–2229
8. Kari, J.: The Nilpotency Problem of One-dimensional Cellular Automata. SIAM Journal on Computing **21** (1992) 571–586
9. Sutner, K.: Cellular automata and intermediate degrees. Theoretical Computer Science **296** (2003)

Coloring Random 3-Colorable Graphs with Non-uniform Edge Probabilities[*]

Ulrik Brandes and Jürgen Lerner[**]

Department of Computer & Information Science, University of Konstanz
lerner@inf.uni-konstanz.de

Abstract. Random 3-colorable graphs that are generated according to a $G(n, p)$-like model can be colored optimally, if $p \geq c/n$ for some large constant c. However, these methods fail in a model where the edge-probabilities are non-uniform and not bounded away from zero. We present a spectral algorithm that succeeds in such situations.

1 Introduction

Graph coloring [9] is one of the central problems in graph theory and combinatorics. A *(proper) graph coloring* is the assignment of colors to vertices so that adjacent vertices are always colored differently. The problem of coloring graphs with the minimum number of colors is of large theoretical interest. Furthermore, efficient coloring algorithms are important for applications, as many practical problems can be formulated as graph coloring problems. However, even if it is known that a graph G is k-colorable, it is $\mathcal{NP}$-hard to properly color G with k colors, for any fixed $k \geq 3$ [6].

Much research has focused on k-coloring random k-colorable graphs with high probability [11,4,15,1,5,12], see [10] for a survey on random graph coloring. (We say that an algorithm succeeds *with high probability (w. h. p.)* if its failure probability tends to zero as the input size tends to infinity.) There are several models for random k-colorable graphs, all of which have the property in common, that every possible edge (i. e., every pair of differently colored vertices) is included in a sampled graph with non-zero probability.

In this paper we propose a more general model for 3-colorable graphs, where there is no lower bound on the edge probabilities. We show that the algorithms from [1,12] can not color graphs from this model and present a more general spectral algorithm that can cope with these distributions. The assumptions that we need for our algorithm are simultaneously more restrictive and more general than those for known algorithms. Thus, we provide an alternative description for random graphs that are easy to color. We believe that our ideas have similar implications for other spectral algorithms (e. g., [12]) that recover "hidden" combinatorial objects (like cliques, independent sets, or minimum cuts) in random graphs.

[*] Research supported by DFG under grant Br 2158/2-3.

[**] Corresponding author.

Our paper is organized as follows. In Sect. 2, we review known models for random 3-colorable graphs, propose a generalization of these models, and present our algorithm and the assumptions under which it succeeds. Section 3 presents general observations regarding planted partitions and spectral algorithms. The ideas developed there are used in Sect. 4 to prove our main theorem. In Sect. 5, we provide the proofs that traditional methods do not work on our generalized model and Sect. 6 outlines implications for related problems.

Notation. If A is a matrix, then A_{uv} denotes the uv'th entry of A. In this paper, the rows and columns of matrices are often understood as being indexed by the vertices of a graph. We frequently use the notation $A(v)$ to denote the column of A that is indexed by the vertex v. The *transpose* of a matrix A is denoted by A^{T} and is defined by $A^{\mathrm{T}}_{uv} = A_{vu}$.

For vectors $v \in \mathbb{R}^n$ we use the *Euclidean norm*, defined by $\|v\|^2 = \sum_{i=1}^{n} v_i^2$. The *distance* between two vectors u and v is $\|u - v\|$. We recall the definition of two matrix norms: the *2-norm*

$$\|A\|_2 = \max_{\|v\|=1} \|A(v)\| \ ,$$

and the *Frobenius norm* $\|A\|_F^2 = \sum_{u,v=1}^{n} A_{uv}^2$. More background on linear algebra is in [13,8].

2 Background and Results

2.1 Previous Models and Known Results

We review first two random graph models for 3-colorable graphs. Let r be a positive integer and p a real number, $0 \leq p \leq 1$. The random graph model $\mathcal{G}(r, p, 3)$ is a probability distribution for graphs on $n = 3r$ vertices, partitioned into three color classes of size r. The edges between vertices from different color classes are included independently with probability p. The best result for this model (i. e., the algorithm that works for the smallest non-trivial p) is from Alon and Kahale [1], who gave a spectral algorithm that (w. h. p.) 3-colors graphs from $\mathcal{G}(r, p, 3)$, if $p \geq c/n$, for a sufficiently large constant c. McSherry [12] described a different spectral algorithm for a more general problem that (w. h. p.) 3-colors graphs from $\mathcal{G}(r, p, 3)$, if $p \geq c \log^3(n)/n$.

It has been pointed out (compare [14]), that random graphs from $\mathcal{G}(r, p, 3)$ have very special properties that graphs encountered in applications usually do not have. It is more demanding to design algorithms for graph models that mediate between the uniformly structured graphs from $\mathcal{G}(r, p, 3)$ and worst-case instances. One possibility to generate such graphs are the so-called *semi-random* graph models. In the semi-random model $\mathcal{G}_S(r, p, 3)$, first a "true random" graph is drawn from $\mathcal{G}(r, p, 3)$, then an adversary can decide to introduce additional edges between vertices from different color classes. While, at a first glance, it seems to help an algorithm if more bi-colored edges are introduced, this is not the case. The fact that the random structure of the graph from $\mathcal{G}(r,p,3)$ is spoiled counts more than the benefit from the additional edges. Feige and Kilian [5] showed that there is a polynomial time algorithm that optimally colors almost all graphs from $\mathcal{G}_S(r,p,3)$ if p is as large as $p \geq (1+\varepsilon)3\log n/n$, for every $\varepsilon > 0$. The algorithm from [5] is not based on spectral methods. Instead it uses semidefinite programming, followed by several sophisticated post-processing steps.

It should be noted that graphs from $\mathcal{G}_S(r,p,3)$ have a substantial subgraph that is a random graph from $\mathcal{G}(r,p,3)$. The adversary is only allowed to add more edges and cannot force any pair of differently colored vertices to be non-adjacent. In this paper, we consider a different probability distribution, where there is no lower bound on the edge-probabilities.

2.2 A Generalization of $\mathcal{G}(r,p,3)$

To generalize the random graph model $\mathcal{G}(r,p,3)$, consider the matrix $A^{(p)}$ in the lefthand-side of (1). (The nine blocks of $A^{(p)}$ are understood as being constant $r \times r$ blocks.) It is easy to see that the sampling process from $\mathcal{G}(r,p,3)$ can be described as follows: construct a graph on $n = 3r$ vertices, where an edge $\{u,v\}$ is introduced with probability equal to $A^{(p)}_{uv}$. The matrix $A^{(p)}$ is the expected adjacency matrix of the distribution $\mathcal{G}(r,p,3)$.

$$A^{(p)} = \left[\begin{array}{ccc|ccc|ccc} 0 & \cdots & 0 & p & \cdots & p & p & \cdots & p \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & p & \cdots & p & p & \cdots & p \\ \hline p & \cdots & p & 0 & \cdots & 0 & p & \cdots & p \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ p & \cdots & p & 0 & \cdots & 0 & p & \cdots & p \\ \hline p & \cdots & p & p & \cdots & p & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ p & \cdots & p & p & \cdots & p & 0 & \cdots & 0 \end{array}\right] \qquad A[XYZ] = \begin{bmatrix} [0] & X & Y \\ X^{\mathrm{T}} & [0] & Z \\ Y^{\mathrm{T}} & Z^{\mathrm{T}} & [0] \end{bmatrix} \qquad (1)$$

The fact that the diagonal blocks of $A^{(p)}$ are zero ensures that graphs from $\mathcal{G}(r,p,3)$ are 3-colorable. The off-diagonal blocks of $A^{(p)}$ describe the expected adjacency structure between two different color classes. In $\mathcal{G}(r,p,3)$ this structure is uniform. Every vertex has the same probability to connect to every other differently colored vertex. We generalize the model $\mathcal{G}(r,p,3)$ by allowing arbitrary adjacency structure between different color classes.

Definition 1. *Let r be an integer and $n = 3r$. Further, let X, Y, and Z, be arbitrary real $r \times r$ matrices whose entries are between zero and one and let $A = A[XYZ]$ be defined as in the righthand-side of (1). A graph drawn from the probability distribution $\mathcal{G}(A)$ is a graph on n vertices, where a set of two vertices $\{u,v\}$ is independently chosen to be an edge with probability A_{uv}. The matrix A is the expected adjacency matrix for the distribution $\mathcal{G}(A)$.*

As an example, the distribution $\mathcal{G}(r, p, 3)$ is equivalent to $\mathcal{G}(A^{(p)})$.

The restrictions on the form of A in Def. 1 ensure only that every graph drawn from $\mathcal{G}(A)$ admits a proper 3-coloring whose color classes are all of size r. In particular, the problem of 3-coloring graphs from $\mathcal{G}(A)$ includes the problem of 3-coloring graphs that are 3-colorable with equally sized color classes. Since this problem is $\mathcal{NP}$-complete in general we cannot hope to develop an algorithm that works for all A.

The distribution $\mathcal{G}(A)$ is obviously much more general than $\mathcal{G}(r, p, 3)$. It is simultaneously more restrictive and more general than the semi-random model $\mathcal{G}_S(r, p, 3)$: In $\mathcal{G}(A)$ we do not allow for an adversary to add edges to a sampled graph. On the other hand, in $\mathcal{G}_S(r, p, 3)$, each possible (bi-colored) edge is included with probability *at least* p (independent on the adversary's decisions), whereas in $\mathcal{G}(A)$, the structure of A can force large sets of possible edges to be not included. Thus, the model $\mathcal{G}(A)$ is an alternative possibility to mediate between the uniformly structured graphs from $\mathcal{G}(r, p, 3)$ and worst-case instances.

2.3 Main Results

Our main contribution is a spectral algorithm that 3-colors (w. h. p.) graphs from $\mathcal{G}(A)$ if the matrices X, Y, and Z are regular with a common degree and if the spectral properties of A "do not obfuscate" the planted 3-coloring. In particular, our algorithm succeeds for many matrices X, Y, and Z for which the algorithms from [1,12] do not work. The algorithm is presented below and gets the $n \times n$ adjacency matrix $\hat{A}$ of a sampled graph as input.

```
Spectral 3-Coloring Algorithm($\hat{A}$)
  1. Compute $d = \sum_{i,j=1}^{n} \hat{A}_{ij}/(2n)$.
  2. Compute (orthonormalized) eigenvectors $\{v_1, v_2, v_3\}$ of $\hat{A}$ associated to
     those eigenvalues that have the smallest distance to $2d$, $-d$ and $-d$.
  3. Let $P$ be the $3 \times n$ matrix whose rows are the $v_i$ and compute $\hat{S} = P^{\mathrm{T}} P$.
  4. Compute the pairwise distances of vertices according to the distance
     between their columns in $\hat{S}$.
  5. Successively join vertices with the smallest distance until three color
     classes are left.
```

See, e. g., [8] for the efficient computation of eigenvectors. Of course, the computed 3-coloring is not necessarily proper for $\hat{A}$. In the following we clarify the assumptions under which the above algorithm succeeds with high probability.

A matrix is called *regular* (of *degree* d) if the sum of every row and column is equal to d. The first assumption we need for our algorithm is that X, Y, and Z must be regular with a common degree.

We turn our attention to the spectral properties of A: If X, Y, and Z are regular of degree d, then (see Theorem 6) A has the three eigenvalues

$$\lambda_{i_1} = 2d, \text{ and } \lambda_{i_2} = \lambda_{i_3} = -d \qquad (i_2 \neq i_3) \ . \tag{2}$$

It is crucial for our algorithm that the other eigenvalues of A are separated from those specific eigenvalues. The *separation* $\mathrm{sep}_3(A)$ of the planted 3-coloring in A is defined to be the minimal distance from λ_{i_1}, λ_{i_2}, and λ_{i_3} to any other eigenvalue of A. We define the *variance* of the distribution $\mathcal{G}(A)$ to be $\sigma^2 = \max_{u,v}(A_{uv} - A_{uv}^2)$ (i. e., the maximal variance of individual entries). The variance is bounded by $1/4$ and goes to zero if all entries of A go either to zero or to one. For instance, for the distribution $\mathcal{G}(r,p,3)$, if $p = c/n$ (that is the smallest p for which the algorithm from [1] is guaranteed to work), then the variance decreases linearly in n, i. e., σ^2 is in $\mathcal{O}(1/n)$. Our main result is the following.

Theorem 1. *Let X, Y, and Z be regular matrices of degree d and $A = A[XYZ]$. Let σ^2 be the variance of $\mathcal{G}(A)$ and assume that $\mathrm{sep}_3(A)$ is in $\omega(n\sigma)$ and that $\sigma^2 \gg (\log^6 n)/n$. Then, the* `Spectral 3-Coloring Algorithm` *properly 3-colors graphs from $\mathcal{G}(A)$, with high probability.*

Theorem 1 is proved in Sect. 4.

As a corollary, we get an algorithm that 3-colors (with probability one) a given graph with adjacency matrix $A[XYZ]$ assumed that X, Y, and Z are regular of common degree and that the separation of the planted 3-coloring is not zero.

Corollary 1. *Let X, Y, and Z be regular $\{0,1\}$ matrices of degree d and assume that $\mathrm{sep}_3(A[XYZ]) \neq 0$. Then, the* `Spectral 3-Coloring Algorithm` *properly 3-colors the graph G with adjacency matrix $A[XYZ]$.*

Interpretation of assumptions. To interpret the assumptions that we make in our theorems, we note first that also the traditional model $\mathcal{G}(r,p,3)$ implicitly makes assumptions on both, the regularity of the block matrices and the separation of certain eigenvalues. The specific form of the matrix $A^{(p)}$ in (1) ensures in particular that the submatrices are regular of degree $d = rp$. In this paper the property of being constant is relaxed to that of being regular. From the point of view of the coloring this means that vertices are no longer required to have the same probability to connect to all vertices of different color, but they are only required to have the same expected number of neighbors of each different color.

Turning to the assumption on the separation, we remark that the specific form of the expected adjacency matrix $A^{(p)}$ implies that $A^{(p)}$ has the three eigenvalues $2d$, $-d$ and $-d$, whereas all other eigenvalues are zero and thus well-separated from the aforementioned. Both previous results [1,12] use this observation and the fact that the random deviation from the expected adjacency matrix has w. h. p. eigenvalues in $\mathcal{O}(\sqrt{d})$. Thus, an assumption on the separation of eigenvalues is also made when assuming that graphs are drawn from the standard model $\mathcal{G}(r,p,3)$. We note however that the assumption on the separation that we make in our paper in not competitive to that of [1,12] when applied to the specific model $\mathcal{G}(r,p,3)$. Currently it is unclear whether the post-processing steps of [1] could be adapted to the more general model $\mathcal{G}(A)$. Similarly, [12] uses the fact that vertices that are in the same color class have identical columns in $A^{(p)}$. Since this is no longer true for our model, it is unclear whether the same bounds could be derived.

2.4 Insufficiency of Traditional Methods

We show here that our algorithm can solve many instances that can not be handled by previous spectral algorithms, e. g., [1,12]. The proofs of the following two Theorems are deferred to Sect. 5.

Theorem 2. *For arbitrary large r, there are $r \times r$ matrices X, Y, and Z such that graphs from $\mathcal{G}(A[XYZ])$ are not colored properly by the algorithms in [1] and [12], but the* `Spectral 3-Coloring Algorithm` *succeeds on these graphs.*

The matrices A that appear in the proof of Theorem 2 also seem to yield instances of $\mathcal{G}(A)$ for which the algorithm from [5], which is designed for the semi-random graph model, does not work. Since this algorithm consists of many randomized sub-procedures, it is more complicated to provide an example of $\mathcal{G}(A)$ for which it fails surely (or with high probability). However, we can show that the proofs in [5] do not generalize to $\mathcal{G}(A)$: A graph G (with a planted coloring) is said to have the *k-collision property*, if for every set U of equally colored vertices and every set T of vertices that are colored differently than those of U, such that $|T|$, $|U| \geq k$, there is an edge in G joining U and T. It is proved in [5] (by translating Lemma 6 of [5] to the graph coloring problem, compare Section 3 of [5]), that semi-random graphs G have with high probability the k-collision property for $k = \frac{2n \log \log n}{c \log n}$. This does not hold for $\mathcal{G}(A)$. In particular the proofs in [5] do not generalize to $\mathcal{G}(A)$.

Theorem 3. *For arbitrary large r, there are $r \times r$ matrices X, Y, and Z, such that graphs from $\mathcal{G}(A[XYZ])$ do not have the k-collision property for any k that is in $o(n)$, but are properly colored by the* `Spectral 3-Coloring Algorithm`.

3 Methodology

The expected adjacency matrix $A^{(p)}$ for the distribution $\mathcal{G}(r, p, 3)$ (see (1)) is so convenient for traditional spectral algorithms since vertices from the same color class have identical columns (neighborhoods) in $A^{(p)}$. Therefore, projecting vertices to the column space of $A^{(p)}$ (that space is spanned by those three eigenvectors that have non-zero eigenvalues) trivially reveals the classes. Moreover, this spectral projection is stable to random noise (given certain assumptions) and the algorithm succeeds also on sampled graphs.

This approach fails for the more general model $\mathcal{G}(A)$. Consider the expected adjacency matrix $A = A[XYZ]$ that arises if the off-diagonal blocks are equal to the matrix shown in (3) and the corresponding graph in Fig. 1(*left*).

$$X = Y = Z = \begin{bmatrix} 0 & 0 & p & p \\ 0 & 0 & p & p \\ p & p & 0 & 0 \\ p & p & 0 & 0 \end{bmatrix} \tag{3}$$

Two vertices that are colored the same may have very different (even disjoint) neighborhoods. In particular, projecting vertices to the column space of A does not reveal the color classes. Equation (3) and Fig. 1 indicate how to construct distributions $\mathcal{G}(A)$ for which traditional spectral methods [1,12] fail: introduce large (i. e., of linear size) sub-blocks of X, Y, Z that are zero, i. e., prohibit edges between large subsets of differently colored vertices. To cope with the distribution $\mathcal{G}(A[XYZ])$ we have to apply a different projection.

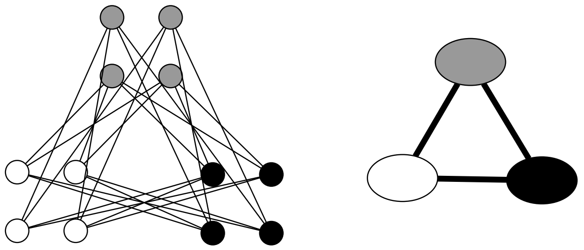


Fig. 1. *Left:* Small example of a non-uniform expected adjacency structure, defined by the block-matrices shown in (3). Edges have weight p. Every white vertex has exactly two black neighbors. *Right:* Quotient induced by the coloring. Edges have weight $2p$.

We represent a vertex k-coloring by a real $k \times n$ matrix P, called the *characteristic matrix* of the coloring, defined by

$$P_{\ell v} = \begin{cases} 1/\sqrt{r} & \text{if vertex } v \text{ is colored } \ell \text{ and } r \text{ is the size of the color class } \ell, \\ 0 & \text{if vertex } v \text{ is not colored } \ell. \end{cases}$$

The characteristic matrix P of the planted 3-coloring projects vertices to 3-dimensional space, such that vertices are mapped to the same point if and only if they are equally colored. Thus, P could be used to determine the planted coloring—we just need a method that identifies the correct P (or a good approximation of it), given only a sample of the distribution.

To derive such a method we observe that, if the block matrices are regular, then the planted 3-coloring satisfies the property of the following definition:

Definition 2. *A coloring is called* structural *(for a symmetric matrix A) if its characteristic matrix P satisfies*

$$\forall u, v \in V \colon P(u) = P(v) \Longrightarrow PA(u) = PA(v) \ .$$

That is, a coloring is structural if, whenever two vertices are colored the same, then they have the same number of each color in their neighborhoods. In algebraic and spectral graph theory, structural colorings are known under the names of *equitable partitions*, or *divisors of graphs* [7,3]. For example, the coloring of the graph in Fig. 1(*left*) is structural.

The idea of structural colorings serves only as a guide to find a projection that recovers the planted 3-coloring. The property of being a structural coloring is not robust to random noise. However, it can be shown, that a relaxation of structural colorings is stable. It is noteworthy, that we do not relax the property of being structural but that of being a discrete coloring.

In the remainder of this section we relax the notion of colorings to *projections* and *similarities*, while keeping the property of being structural. In Sect. 4 we show that (w. h. p.) our algorithm computes the appropriate structural similarity for the sampled matrix and recovers the planted 3-coloring.

Structural similarities have been introduced in [2] as a relaxation for role assignments. (Role assignments identify structurally similar vertices in networks.) Here we review some concepts from [2] in a slightly different notation.

Projections and similarities are introduced as relaxations of k-colorings and their associated equivalence relations on the vertex set:

Definition 3. *A real $k \times n$ matrix P with orthonormal rows is called a* projection. *If P is a projection, then the real $n \times n$ matrix $S = P^{\mathrm{T}}P$ is called the* similarity *associated with P. Let, in addition, be A the adjacency matrix of a graph. Then the real $k \times k$ matrix $B = PAP^{\mathrm{T}}$ is called the* quotient *induced by A and P.*

The characteristic matrix of a coloring is a special case of a projection. Projections are more general than colorings, since they allow vertices to be members of several color classes: the v'th column of a projection P is a k-dimensional vector that describes the real-valued membership of vertex v to the k color-classes. The entry S_{uv} of the associated similarity S is the dot-product of the u'th and v'th column of P. Thus u and v have high similarity if they are similarly colored. From an algebraic point of view, a similarity is the orthogonal projection to the row-space of P (compare [2]). If P is the characteristic matrix of a coloring, then the quotient $B = PAP^{\mathrm{T}}$ is the adjacency matrix of the weighted graph that has the k color-classes as vertices and two classes c_1 and c_2 are connected by an edge whose weight is the sum over all edges between c_1 and c_2 divided by $\sqrt{|c_1| \cdot |c_2|}$. For an example, see Fig. 1. The following definition introduces the attribute structural for similarities. It is then noted in Theorem 4 that structural similarities are indeed relaxations of structural colorings.

Definition 4. *Let P be a projection and let S be its associated similarity, then P and S are called* structural *for a matrix A if $SA = AS$.*

Theorem 4 ([2]). *Let P be the characteristic matrix of a vertex coloring $c\colon V \to \{1, \ldots, k\}$. Then, P is a structural projection if and only if c is a structural coloring.* □

The following Theorem provides the link between spectral techniques and structural similarities. Further it shows how similarities that yield a pre-specified quotient can be efficiently computed.

Theorem 5 ([2]). *Let A be a symmetric $n \times n$ matrix, B a symmetric $k \times k$ matrix, P a projection, and S its associated similarity. Then P and S are structural for A if and only if the image of S is generated by eigenvectors of A.*

Furthermore, P and S are structural for A and the induced quotient equals B if and only if those eigenvectors are associated to the eigenvalues of B. □

If P and S are structural, we call the eigenvalues of B *associated* to P and S.

Traditional spectral methods typically chose projections associated to the eigenvalues with the largest absolute values (compare [12]). Structural similarities are not restricted to projecting to the largest eigenvalues but can chose all subsets and thereby can recover partitions in more general situations (as demonstrated in this paper). The second part of Theorem 5 is important for determining which eigenvalues have to be chosen for a specific task.

4 Correctness of the Algorithm

Throughout this section, let $A = A[XYZ]$ be a real $n \times n$ matrix as in the righthand side of (1) and let $\mathcal{G}(A)$ be the associated distribution of 3-colorable graphs (compare Def. 1). Let $\hat{A}$ be the adjacency matrix of a sample drawn from $\mathcal{G}(A)$. Further, let σ^2 be the variance of the distribution and assume that $\sigma^2 \gg (\log^6 n)/n$.

The following theorem states that, for regular X, Y, and Z, there is a structural similarity for A, which reveals the planted 3-coloring. Theorem 6 does not rely on any assumptions on $\operatorname{sep}_3(A)$.

Theorem 6. *Let X, Y, and Z be regular $r \times r$ matrices with degree d. Then, there is a structural similarity S that has $2d$, $-d$, and $-d$ as associated eigenvalues and satisfies for all vertices u and v,*

$$\|S(u) - S(v)\| = \begin{cases} 0 & \text{if } u \text{ and } v \text{ are colored the same, and} \\ \sqrt{2/r} & \text{if } u \text{ and } v \text{ are colored differently.} \end{cases} \tag{4}$$

Proof. Since the matrices X, Y, and Z are regular, the planted 3-coloring is a structural coloring for A. Thus, by Theorem 4 its characteristic matrix P, is a structural projection. Furthermore, the induced quotient $B = PAP^{\mathrm{T}}$ is the adjacency matrix of a triangle whose three edges have weight d. Thus, the associated eigenvalues of P are $2d$, $-d$, and $-d$. Finally, the similarity $S = P^{\mathrm{T}}P$ satisfies (4). □

We show in Theorem 8 that there is a structural similarity $\hat{S}$ for the *sampled* adjacency matrix $\hat{A}$ that is close enough to S. First we have to recall a well-known bound on the eigenvalues of random matrices.

Theorem 7. *Let F be defined by $F = A - \hat{A}$. Then (w. h. p.) it is $\|F\|_2 \leq 4\sigma\sqrt{n}$* [12]. *In particular, the eigenvalues of $\hat{A}$ differ (w. h. p.) from those of A by at most $4\sigma\sqrt{n}$* [13].

Theorem 8. *Let X, Y, and Z be regular $r \times r$ matrices with common degree d and $A = A[XYZ]$. Further, let S be the similarity from Theorem 6 and assume that $\operatorname{sep}_3(A)$ is in $\omega(\sigma\sqrt{n})$. Then, w. h .p. the similarity $\hat{S}$ that is associated to those three eigenvalues of $\hat{A}$ that have the smallest distance to $2d$, $-d$, and $-d$ satisfies $\|\hat{S} - S\|_2 \in \mathcal{O}(\sigma\sqrt{n}/\operatorname{sep}_3(A))$.*

Proof. (The following assertions hold w. h. p. for sufficiently large n.) By the assumption on $\operatorname{sep}_3(A)$ and Theorem 7, there are three well-defined eigenvalues λ_{i_1}, λ_{i_2}, and λ_{i_3} of $\hat{A}$ that have the smallest distance to $2d$, $-d$, and $-d$. Let v_{i_1}, v_{i_2}, and v_{i_3} be three orthonormal eigenvectors of $\hat{A}$, associated to λ_{i_1}, λ_{i_2}, and λ_{i_3}. Let C be the $n \times 3$ matrix whose columns are the v_{i_j}, $j = 1, 2, 3$. We show that $\hat{S} = CC^{\mathrm{T}}$ satisfies the assertions of the theorem.

By Theorem 5, $\hat{S}$ is structural for $\hat{A}$ and λ_{i_1}, λ_{i_2}, and λ_{i_3} are the eigenvalues associated to $\hat{S}$. To show the bound on $\|\hat{S} - S\|$, let $M = C^{\mathrm{T}}\hat{A}C$, B_1 be an $n \times 3$ matrix whose columns span the image of the similarity S, and B_2 be an $n \times (n-3)$ matrix, such that $(B_1 B_2)$ is an orthogonal $n \times n$ matrix. Let F be defined by $\hat{A} = A - F$ and set $L = B_2^{\mathrm{T}} A B_2$. By definition of M and the fact that $\hat{S}$ commutes with $\hat{A}$ ($\hat{S}$ is structural for $\hat{A}$), it is $0 = \hat{A}C - CM$. By the definition of F it follows $FC = AC - CM$. Let δ be the minimal distance between eigenvalues of M and those of L. By the assumptions on the separation $\operatorname{sep}_3(A)$ and Theorem 7, δ is in $\Omega(\operatorname{sep}_3(A))$ and it follows with Theorems V.3.4 and I.5.5 of [13] that

$$\|S - \hat{S}\|_2 \leq \frac{2\|FC\|_F}{\delta} \ .$$

The 2-norm of F is bounded by $4\sigma\sqrt{n}$ (Theorem 7), the Frobenius norm of the $n \times 3$ matrix FC is at most $\sqrt{3}$-times the 2-norm of FC, and the 2-norm of the matrix C (having orthonormal columns) is 1. Thus, the assertion follows with

$$\|FC\|_F \leq \sqrt{3}\|FC\|_2 \leq \sqrt{3}\|F\|_2 \leq 4\sigma\sqrt{3n} \ .$$

□

To determine $\hat{S}$, the degree d of the block matrices has to be estimated:

Lemma 1. *Let X, Y, and Z be regular $r \times r$ matrices with common degree d and $A = A[XYZ]$. Let $\hat{A}$ be the adjacency matrix of a graph drawn from $\mathcal{G}(A)$ and set $\hat{d} = \sum_{i,j=1}^{n} \hat{A}_{ij}/(2n)$. Then, with high probability, $\hat{d} - d$ is in $\mathcal{O}(\log n)$.*

Proof. Follows in a straightforward manner from the Hoeffding bound. □

Proof (of Theorem 1). By Lemma 1 and Theorem 7, the similarity $\hat{S}$, as computed by the algorithm, is the similarity from Theorem 8. (Note that $\hat{S} = P^{\mathrm{T}}P$ is independent on orthogonal transformations on the rows of P like, e. g., permutation or reflexion of eigenvectors.) Let v be any vertex and let S be the similarity from Theorem 6. We have by Theorem 8 that w. h. p.

$$\|S(v) - \hat{S}(v)\| \in o(1/\sqrt{n}) \ .$$

Hence, for two vertices u and v it is (applying Theorem 6)

$$\|\hat{S}(u) - \hat{S}(v)\| \in \begin{cases} o(1/\sqrt{n}) & \text{if } u \text{ and } v \text{ are in the same color class,} \\ \Omega(1/\sqrt{n}) & \text{else} \ . \end{cases}$$

Thus for sufficiently large n the clustering procedure in Step 5 yields exactly the planted color classes. □

Proof (of Corollary 1). Follows from Theorem 1 by considering the zero-variance distribution that assigns probability one to G (and probability zero to any other graph). Following the proofs in Sect. 4, it can be seen that the restriction "with high probability" from Theorem 1 can be dropped in this situation. □

5 Hard Instances for Traditional Methods

Proof (of Theorem 2). The instances for which the algorithm from [1] does not work are essentially a blown-up version of the example in (3) and Fig. 1 with a few added edges. For simplicity we take $p = 1$ in our example. This implies that only one graph has non-zero probability in the distribution. It should be obvious that similar examples with probabilities different from zero or one can be constructed.

Let $r = 2k$ for an integer k. Let H be the graph that is the complete bipartite graph $K_{k,k}$ plus the edges of two cycles of length k, connecting the vertices in the bipartition classes of $K_{k,k}$ (thereby making H non-bipartite). Let $X = Y = Z$ denote the adjacency matrix of H and let $A = A[XYZ]$. Let G be the graph with adjacency matrix A (the unique graph returned by $\mathcal{G}(A)$).

The preprocessing step from [1] is void in this case since G is regular. The last eigenvector v_n has median zero. Further for $t = v_n$, in the first phase of the algorithm from [1] the vertices are colored with only two colors, according to which "bipartition class" they belong to. In particular this coloring is a very bad approximation to the unique proper 3-coloring and it is easy to see that the second and third phase in the proposed algorithm do not overcome this.

Examples of distributions for which the algorithm in [12] does not recover the planted 3-coloring, are quite similar to the one above.

Finally, the above distribution satisfies the assumptions of our theorems and, hence, G can be colored by our algorithm: The matrices X, Y, and Z are d-regular by construction, where $d = k + 2$. By computing the eigenvalues of the complete bipartite subgraphs and applying facts about the eigenvalues of the Kronecker product of matrices (compare [3]), we get that the eigenvalues $2d$, $-d$, and $-d$ have non-zero separation from the others and thus Corollary 1 applies. □

Proof (of Theorem 3). In the example above there are suitable sets U and T of size linear in n such that there is no edge joining U and T. □

6 Concluding Remarks

The ideas of this paper are not restricted to graph coloring. Many heuristics for $\mathcal{NP}$-hard graph partitioning problems (like min-bisection, clique, or independent set) are based on spectral techniques that typically chose projections associated to the eigenvalues with the largest absolute values. McSherry [12] showed that these specific spectral projections recover partitions if vertices in the same class have identical columns (neighborhoods) in the expected adjacency matrix. It seems that the converse is also true: these specific spectral projections recover partitions *only if* vertices in the same class have almost identical neighborhoods in the expected adjacency matrix. We outlined in Sect. 3 that projections associated to eigenvalues that are not necessarily the largest may succeed in more general situations, where vertices from the same class have only same-colored (instead of identical) neighborhoods. It seems to be promising to consider these generalized spectral projections also for the solution of other problems.

References

1. N. Alon and N. Kahale. A spectral technique for coloring random 3-colorable graphs. *SIAM Journal on Computing*, 26:1733–1748, 1997.
2. U. Brandes and J. Lerner. Structural similarity in graphs. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC'04)*, pages 184–195, 2004.
3. D. M. Cvetković, M. Doob, and H. Sachs. *Spectra of Graphs.* Johann Ambrosius Barth, 1995.
4. M. E. Dyer and A. M. Frieze. Fast solution of some random $\mathcal{NP}$-hard problems. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS'86)*, pages 331–336, 1986.
5. U. Feige and J. Kilian. Heuristics for semi-random graph problems. *Journal of Computer and System Sciences*, 63(4):639–671, 2001.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-completeness.* Freeman and Company, 1979.
7. C. Godsil and G. Royle. *Algebraic Graph Theory.* Springer, 2001.
8. G. H. Golub and C. F. van Loan. *Matrix Computations.* John Hopkins University Press, 1996.
9. T. R. Jensen and B. Toft. *Graph Coloring Problems.* Wiley, 1995.
10. M. Krivelevich. Coloring random graphs – an algorithmic perspective. In *Proceedings of the 2nd Colloquium on Mathematics and Computer Science (MathInfo'2002)*, pages 175–195, 2002.
11. L. Kučera. Expected behavior of graph coloring algorithms. In *Lecture Notes in Computer Science* 56, pages 447–451. Springer Verlag, 1977.
12. F. McSherry. Spectral partitioning of random graphs. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS'01)*, pages 529–537, 2001.
13. G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory.* Academic Press, 1990.
14. C. R. Subramanian, M. Fürer, and C. E. V. Madhavan. Algorithms for coloring semi-random graphs. *Random Structures and Algorithms*, 13:125–158, 1998.
15. J. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.

The Kleene Equality for Graphs

Arnaud Carayol and Didier Caucal

IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
{Arnaud.Carayol, Didier.Caucal}@irisa.fr

Abstract. In order to generalize the Kleene theorem from the free monoid to richer algebraic structures, we consider the non deterministic acceptance by a finite automaton of subsets of vertices of a graph. The subsets accepted in such a way are the equational subsets of vertices of the graph in the sense of Mezei and Wright. We introduce the notion of deterministic acceptance by finite automaton. A graph satisfies the Kleene equality if the two acceptance modes are equivalent, and in this case, the equational subsets form a Boolean algebra. We establish that the infinite grid and the transition graphs of deterministic pushdown automata satisfy the Kleene equality and we present families of graphs in which the free product of graphs preserves the Kleene equality.

1 Introduction

In the case of the free monoid, Kleene theorem states the equivalence between acceptance by non-deterministic finite state automata and acceptance by deterministic and complete automata. As a consequence the regular sets of words are closed under complementation and form a Boolean algebra. For arbitrary monoids, the regular subsets (which are the subsets of the monoid accepted by finite automata labeled by elements of this monoid) are closed under union but are not, in general, closed under complementation. However there are well-known examples of monoids for which the regular subsets form a Boolean algebra: the free commutative monoid [GS64], the trace monoid with transitive independence relations [Sak87], the context-free groups [Sén96]... As these monoids are not free, the acceptance by deterministic and complete automata no longer provides the closure under complementation.

The goal of this article is to explain the closure under complementation of the regular subsets of such monoids by a notion of deterministic acceptance by finite state automaton. In order to do so, we consider the acceptance by finite automata of subsets of vertices of a colored graph.

A finite automaton is simply a finite colored graph whose vertices are called states, together with a set of final states. The run of the automaton A on a graph G is a relation between the vertices of the graph G and the states of A. It is the smallest relation ρ such that if a vertex x of G and a state q of A are colored by a same color, then (x, q) also belongs to ρ and such that if (x, p) belongs to ρ and there is a a-labeled edge from x to y in G and a a-labeled transition from p to q in A, then (y, q) belongs to ρ. Intuitively, the colored vertices of the graph act as starting points for the automaton and the colored states in the automaton play the role of initial states. The subset of vertices accepted by A is the set of vertices coupled by ρ with at least one final state. We denote $\mathrm{AFS}(G)$ the set of all subsets of vertices accepted by a finite automaton running on G. The set $\mathrm{AFS}(G)$ is in fact the set of all equational subsets (in the sense of Mezei and Wright [MW67]) of the subset algebra associated to G.

This framework encompasses the case of the monoids previously mentioned when considering their Cayley graphs [Cay78]. Every monoid $\mathcal{M}$ finitely generated by a subset $P \subseteq M$ can be represented by its Cayley graph $\mathcal{C}(\mathcal{M}, P)$ whose vertices are the elements of the monoid and where an edge labeled by $p \in P$ represent the product on the right by p and where the neutral element is colored by ι. For instance, the Cayley graphs of the free monoids over two letters $\{a, b\}$ is the full binary tree labeled by a and b with its root colored by ι. The subsets of the Cayley graph of $\mathcal{M}$ accepted by a finite automaton are the regular subsets of $\mathcal{M}$.

The notion of deterministic acceptance is fairly simple: the run of an automaton is deterministic and complete if it associates one and only one state to each vertex of the graph. The set of all subsets of vertices accepted by A with a deterministic and complete run is denoted $\mathrm{DAFS}(G)$. A graph G is said to satisfy the Kleene equality if $\mathrm{AFS}(G) = \mathrm{DAFS}(G)$. In this case, $\mathrm{AFS}(G)$ is a Boolean algebra.

To substantiate the pertinence of the notion of deterministic acceptance, we show that the Cayley graph of the free commutative monoid with two generators, the rooted graphs of deterministic pushdown automata [MS85] (which contain the Cayley graphs of context-free groups) and the rooted deterministic prefix-recognizable graphs [Cau96] all satisfy the Kleene equality. Finally we provide sufficient conditions for the free product of graphs to preserve the Kleene equality.

2 Preliminaries

The *inverse* of a relation $R \subseteq P \times P$ is $R^{-1} := \{ (q,p) \mid (p,q) \in R \}$. The *image* of $Q \subseteq P$ by R is $R(Q) := \{ p \in P \mid \exists q \in Q, (q,p) \in R \}$. The *product* of two relations R and S is $R \cdot S := \{ (p,r) \mid \exists q \in P, (p,q) \in R \text{ and } (q,r) \in S \}$.

Colored graphs. A *colored graph* G labeled by a finite set Σ and colored by a finite set C is a subset of $(V \times \Sigma \times V) \cup (C \times V)$ for some countable V. The set of *vertices* of G is $V_G := \{ u \in V \mid \exists v \in V, a \in \Sigma, c \in C, (u,a,v) \in G \text{ or } (v,a,u) \in G \text{ or } (c,u) \in G \}$, its set Σ_G of labels is $\{ a \in \Sigma \mid \exists u, v \in V_G, (u,a,v) \in G \}$ and its set C_G of colors $\{ c \in C \mid \exists u \in V_G, (c,u) \in G \}$. If (u,a,v) belongs to G, we will say that there is an a-labeled edge from u to v. If (c,u) belongs to G, we say that u is colored by c.

A graph H is a *subgraph* of G if $H \subseteq G$ and it is a *covering subgraph* if $V_H = V_G$.

A *path* in G is a sequence $u_0 a_1 u_1 \dots a_n u_n \in V_G(\Sigma_G V_G)^*$ such that for all $i \in [1, n-1]$, $(u_i, a_{i+1}, u_{i+1}) \in G$. A graph G is *accessible* from its colors if for all $x \in V_G$, there exists a path from some colored vertex $i \in V_G$ to x.

A graph G is *rooted* if there exists a color c and a vertex r called the *root* colored by c such that r is the only vertex colored by c and every vertex is reachable from r.

A graph G is *deterministic* if for all $(u, a, v) \in G$, if $(u, a, v') \in G$ then $v = v'$ and for all $(c, u) \in G$, if $(c, u') \in G$ then $u = u'$. A graph G is *(source) complete* if for all $u \in V_G$ and $a \in \Sigma$, there exists $v \in V_G$ such that $(u, a, v) \in G$.

A *morphism* φ from a graph G to a graph H is a mapping from V_G to V_H such that $(u, a, v) \in G$ implies $(\varphi(u), a, \varphi(v)) \in H$ and $(c, u) \in G$ implies $(c, \varphi(u)) \in H$.

For every graph G with $C_G \cap \Sigma_G = \emptyset$, we define the *subset algebra* of G which is a unary algebra over the signature $C_G \cup \Sigma_G$ where the symbols in C_G are constants and the symbols in Σ_G are unary functions. Its carrier is 2^{V_G} and symbols in $C_G \cup \Sigma_G$ are interpreted in the following way: for all $c \in C_G$, $\mathbf{c} = \{\, v \in V_G \mid (c, v) \in G \,\}$ and for all $a \in \Sigma$ and $M \subseteq V_G$, $\mathbf{a}(M) = \{\, v \mid \exists u \in M, (u, a, v) \in G \,\}$.

Monoids and their Cayley graphs. A *monoid* $\mathcal{M} = (M, \cdot)$ is given by a set M and an associative product $\cdot$ admitting a neutral element $1_M \in M$. The product is extended to subsets of M by taking for all $P, Q \subseteq M$, $P \cdot Q := \{\, p \cdot q \mid p \in P \text{ and } q \in Q \,\}$. For all $P \subseteq M$, $P^* = \bigcup_{i \in \mathbb{N}} P^i$ where $P^0 = \{1_M\}$ and $P^{i+1} = P^i \cdot P$. The set of all regular subsets of $\mathcal{M}$ denoted by $\mathrm{Reg}(\mathcal{M})$ is the smallest set containing the finite sets and closed under union, concatenation and the star operation.

A monoid $\mathcal{M} = (M, \cdot)$ is *finitely generated* by a finite subset P of M if $M = P^*$. Its Cayley graph $\mathcal{C}(M, P)$ is labeled by P and colored by ι and defined by:

$$C(M, P) := \{\, (m, p, m \cdot p \mid m \in M \text{ and } p \in P \,\} \cup \{(\iota, 1_M)\}.$$

3 Non-deterministic and Deterministic Finite State Acceptance

In Subsection 3.1, we present the equational subsets of vertices of a graph as the subsets accepted by finite state automata running on the graph. In Subsection 3.2, we introduce the notion of subsets of vertices of a graph deterministically accepted by finite automata and state the generalization of the Kleene equality. Finally in Subsection 3.3, we compare this new notion with the well-known notion of recognizable subsets [MW67].

3.1 Non-deterministic Finite State Acceptance

A *finite automaton* A is a finite colored graph whose vertices are called *states* together with a finite set of final states $F \subseteq V_A$.

The *run* of a finite automaton A on a graph G is the smallest relation $\rho \subseteq V_G \times V_A$ satisfying:

- for all $c \in C_G \cap C_A$, all $x \in V_G$ and $p \in V_A$, if $(c, x) \in G$ and $(c, p) \in A$ then $(x, p) \in \rho$,
- for all $a \in \Sigma_G \cap \Sigma_A$, $x, y \in V_G$ and $p, q \in V_A$, if $(x, p) \in \rho$, $(x, a, y) \in G$ and $(p, a, q) \in A$ then $(y, q) \in \rho$.

The subset of V_G accepted by A is $\rho^{-1}(F)$. The subsets of G accepted by some finite automaton will be called the subsets accepted by finite state (AFS for short) and will be designated by AFS(G).

Intuitively, the colored vertices of the graph play the role of starting point for the automaton. We say that the automaton goes through an a-labeled edge $(x, a, y) \in G$ if for some edge $(p, a, q) \in A$, we have $(x, p) \in \rho$ and $(y, q) \in \rho$.

These subsets are the equational subsets (as originally defined in [MW67]) of the subset algebra of G. In fact for all $q \in V_A$, the sets $\rho^{-1}(\{q\})$ are the smallest solution (for the inclusion) of the following finite set of equations on the subset algebra associated to G:

$$X_q = \bigcup_{(c,q)\in A} \mathbf{c} \quad \cup \bigcup_{(p,a,q)\in A} \mathbf{a}(X_p).$$

In [Cou89], a characterization of equational subsets by finite automata is provided. Our definition differs slightly in the definition of the run on the automaton.

For all finite automata A and B with $V_A \cap V_B = \emptyset$, the run of $A \cup B$ on G is the union of the run of A on G and of the run of B on G. Hence $AFS(G)$ is closed under union.

Proposition 1. *For all G,* AFS(G) *is closed under union and contains $\emptyset$.*

It is well-known that AFS(G) is not in general closed under complementation. In fact, remark that V_G does not, in general, belongs to AFS(G): if a vertex x of G is not accessible from any colored vertex, then it does not belong to any set in AFS(G). As we investigate cases for which AFS(G) is a Boolean algebra, it is reasonable to assume that the graphs under consideration are accessible from their colors. In the following, we will always assume that the graphs under consideration are accessible from their colors.

Example 1. Consider the graph Stacks associated to pushdown stacks over the alphabet $\Gamma = \{a, b\}$:

$$\begin{aligned} \text{Stacks} := \quad & \{\, (u, x, ux) \mid x \in \{a, b\} \text{ and } u \in \Gamma^* \,\} \\ \cup\, & \{\, (ux, \bar{x}, u) \mid x \in \{a, b\} \text{ and } u \in \Gamma^* \,\} \cup \{(\iota, \varepsilon)\}. \end{aligned}$$

The vertices are the stacks over Γ (which are simply words in Γ^*) and the edges represent the basic operations on stacks: an edge labeled by a (resp. b) represents the push of the letter a (resp. b) on top of the stack and an edge labeled by $\bar{a}$ (resp. $\bar{b}$) the removal of the top most letter of the stack if it is an a (resp. b). The empty stack is colored by ι.

Every set $V \in$ AFS(Stacks) can be seen as the set of stack contents appearing in a final reachable configurations of some pushdown automaton. In fact if we omit the input alphabet, a pushdown automaton is simply a finite automaton labeled by the operations $\{a, b, \bar{a}, \bar{b}\}$. In [Büc64], Büchi showed that these sets are regular. Hence AFS(Stacks) = Reg(Γ^*) is a Boolean algebra.

Figure 1 presents the run ρ of a finite automaton A on Stacks. The set of states of A associated by ρ to a vertex x of Stacks are written in boldface next to x. If we take $\{r_1, l_1\}$ as set of final states for A, A accepts the set $\{aa, bb\}^* \cdot \{a, b\}$.

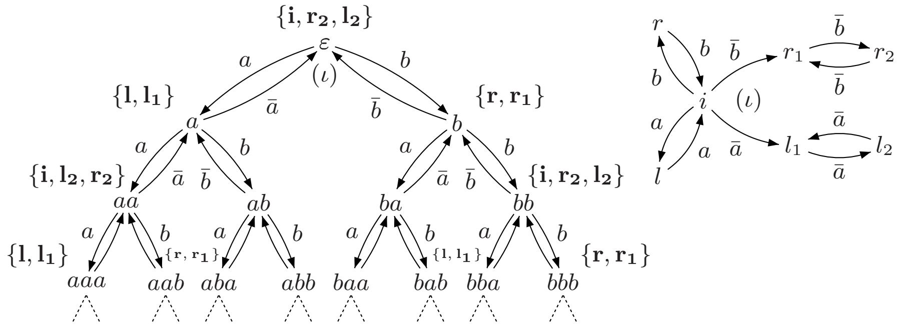


Fig. 1. The run (on the left) on Stacks of a finite automaton A (on the right)

Other meaningful examples of graphs for which AFS(G) is a Boolean algebra are provided by considering the Cayley graphs of finitely generated monoids. The sets accepted by finite automata running on the Cayley graph of a finitely generated monoid are the regular subsets of this monoid.

Proposition 2. *For any monoid $\mathcal{M}$ finitely generated by P,*

$$\mathrm{AFS}(\mathcal{C}(\mathcal{M}, P)) = \mathrm{Reg}(\mathcal{M}).$$

Example 2. A first simple example of Cayley graph is the Cayley graph Δ_2 of the free monoid $\{a, b\}^*$ presented in Figure 2. Consider now the free commutative monoid with two generators $(\mathbb{N}^2, +)$ where $+$ designates the componentwise addition. Its Cayley graph Grid with respect to the generating set $\{(0, 1), (1, 0)\}$ is the infinite grid depicted in Figure 2. By Proposition 2, the subsets accepted by finite automata on Grid are the regular subsets of $(\mathbb{N}^2, +)$ which are also known as the semi-linear sets of $\mathbb{N}^2$. From [GS64], we know that AFS(Grid) is a Boolean algebra.

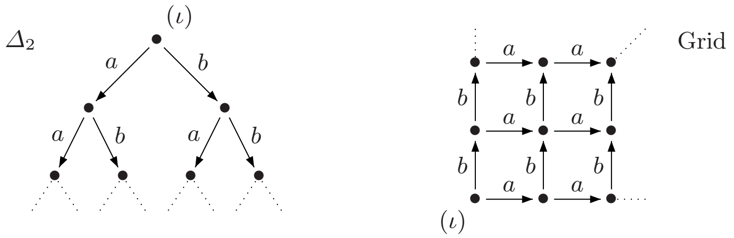


Fig. 2. The Cayley graph Δ_2 of the free monoid $\{a, b\}^*$ and the Cayley graph Grid of $(\mathbb{N}^2, +)$ with $a = (1, 0)$ and $b = (0, 1)$

By a standard powerset construction, a non deterministic automaton A can be transformed into a deterministic and complete automaton accepting the same subset.

Proposition 3 ([MW67]). *For all graph G, every set in* $\mathrm{AFS}(G)$ *is accepted by a deterministic and complete automaton.*

As shown in Example 1, a deterministic automaton does not necessarily have a deterministic behavior: its run can assign several states to the same vertex. We will say that such a run is *non deterministic.*

3.2 Deterministic Finite State Acceptance

We lift the notion determinism and completeness from the automaton to the behaviour of the automaton : its run.

Definition 1. *The run ρ of a finite automaton A on a graph G is said to be deterministic and complete if for every vertex $x \in V_G$, there exists one and only one state $p \in V_A$ such that $(x,p) \in \rho$.*

In other terms, ρ is a mapping. In this case, we will adopt the functional notation and write $\rho(x) = p$ instead of $(x,p) \in \rho$.

The set of all subsets of G accepted by a finite automaton having a deterministic and complete run on G is written $\mathrm{DAFS}(G)$. In the following we will simply say that these subsets of vertices are deterministically accepted. By definition, for all graph G, $\mathrm{DAFS}(G)$ is included in $\mathrm{AFS}(G)$.

Note that every set $V \in \mathrm{DAFS}(G)$ is deterministically accepted by a deterministic automaton. However note that contrary to what we obtained for $\mathrm{AFS}(G)$, we can no longer assume that this automaton is complete (see Subsection 3.3 for a discussion on this fact).

As the automaton is not necessarily complete, its run on a graph G induces a subgraph of G obtained by only keeping the edges of G borrowed by the automaton. For any graph G and any finite automaton A with a deterministic run ρ on G, the graph of the run ρ is the graph G_ρ defined by: $G_\rho := \{\, (u,a,v) \in G \mid (\rho(u), a, \rho(v)) \in A \,\}$. As ρ is complete, G_ρ is a covering subgraph of G.

Example 3. Consider the finite automaton A of Figure 3. Its run on Grid (cf. Figure 2) is deterministic and complete. The graph of its run Grid_ρ is represented in Figure 3. If we take $\{r, s\}$ as set of final states for A, A accepts the diagonal of the grid $\{\, (n,n) \mid n \in \mathbb{N} \,\}$.

For any set $V \in \mathrm{DAFS}(G)$, there exists a finite automaton A with a set $F \subseteq V_A$ of final states and a deterministic run ρ on G such that $V = \rho^{-1}(F)$. As ρ is a mapping, $V_G \setminus V = \rho^{-1}(V_A \setminus V_F)$ and it follows that $\mathrm{DAFS}(G)$ is closed under complementation.

Proposition 4. *For all graph G,* $\mathrm{DAFS}(G)$ *is closed under complementation.*

However $\mathrm{DAFS}(G)$ is not in general closed under union and intersection. For example consider the graph G and the two automata A and B presented in Figure 4. These two automata have a deterministic run on G and by taking f as unique final state, they accept $\{p\}$ and $\{s\}$ respectively. However $\{p, s\}$ does not

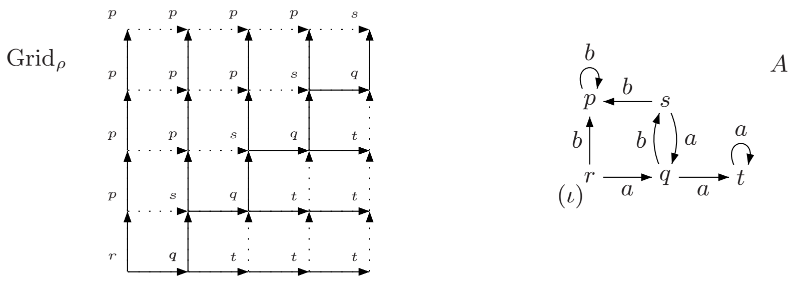


Fig. 3. A finite automaton A with a deterministic run ρ on Grid and the graph of its run Grid$_\rho$

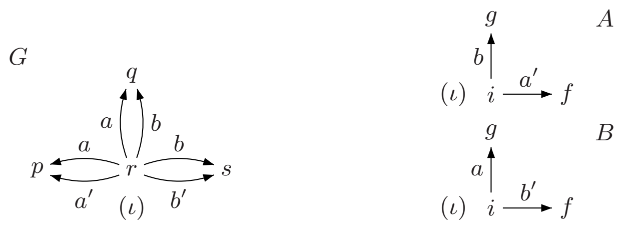


Fig. 4. Example of graph G for which DAFS(G) is not closed under union

belong to DAFS(G). In fact for all deterministic and complete run ρ, we have either $\rho(q) = \rho(p)$ or $\rho(q) = \rho(s)$.

If non deterministic and deterministic acceptances are equivalent on G, we say that G satisfies the Kleene equality.

Definition 2. *A graph G satisfies the* Kleene equality *if* AFS(G) = DAFS(G).

As by Proposition 1, AFS(G) is closed under union and contains the empty set and by Proposition 4, DAFS(G) is closed under complementation, it follows that AFS(G) is a Boolean algebra.

Proposition 5. *For any G, if* AFS(G) = DAFS(G) *then* AFS(G) *is a Boolean algebra.*

The Kleene equality states the equivalence between non-deterministic and deterministic acceptance by finite automata. A stronger requirement is that every non-deterministic run can be determinized.

Definition 3. *For any graph G, G satisfies the* strong Kleene equality *if for every finite automaton A with a run ρ_A on G, there exists a finite automaton B with a deterministic and complete run ρ_B on G and a relation $\eta \subseteq V_B \times V_A$ such that $\rho_A = \rho_B \cdot \eta$.*

Intuitively, this means that for any finite automaton A there exists an automaton B with a deterministic run of G such that the set of states associated to a vertex

x of G is entirely characterized by the unique state $\rho_B(x)$ associated by B to x: $\rho_A(x) = \eta(\rho_B(x))$.

In particular, if F_A is the set of final states of A, by taking $F_B = \eta^{-1}(F_A)$ as set of final states for B, B accepts the same subset as A. Hence, if a graph satisfies the strong Kleene equality, it satisfies the Kleene equality.

Example 4. Consider the graph Stacks and the finite automaton A presented in Example 1. The automaton B presented in Figure 5 has a deterministic and complete run ρ_B on Stacks and accepts the same language as A if we take $\{p, s\}$ as final states. Moreover taking $\eta = \{(r, i), (r, r_2), (r, l_2), (s, l), (s, l_1), (p, r), (p, r_1)\}$, we have $\rho_A = \rho_B \cdot \eta$. In fact as Stacks is a rooted and deterministic pushdown transition graph, we will prove in Subsection 4.2 that it satisfies the strong Kleene equality.

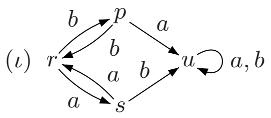


Fig. 5. An automaton B with a deterministic and complete run on Stacks (cf. Ex. 1)

3.3 Comparison with Recognizable Subsets

The notion of recognizable subsets of a monoid was introduced by Eilenberg. Mezei and Wright extended this notion to algebras in [MW67]. For a graph G, a subset of vertices $V \subseteq V_G$ is recognizable if there exists a deterministic and complete automaton A with $\Sigma_A = \Sigma_G$ and $C_A = C_G$ and a morphism φ from G to A such that $V = \varphi^{-1}(\varphi(V))$. We write Rec($G$) the set of all recognizable subsets of vertices of G.

It is well-known that for any graph G, Rec(G) is a Boolean algebra. In our setting, this notion can be captured by considering deterministic and complete automaton having a deterministic run.

Proposition 6. *For all graph G, a subset V of V_G is recognizable if and only if it is deterministically accepted by a deterministic and complete finite automaton A with $\Sigma_A = \Sigma_G$ and $C_A = C_G$.*

A direct consequence of this characterization is that for all graph G, we have $\text{Rec}(G) \subseteq \text{DAFS}(G) \subseteq \text{AFS}(G)$. In the case of the free monoid $\{a, b\}^*$ (whose Cayley graph is Δ_2 presented in Figure 2), it is well-known that $\text{Rec}(\Delta_2) = \text{AFS}(\Delta_2)$. Hence Δ_2 satisfies the Kleene equality. In general, Rec(G) is strictly included in DAFS(G). Consider for example the graph Grid presented in Figure 2: we have seen in Example 3 that $\{\, (n, n) \mid n \in \mathbb{N} \,\}$ belongs to DAFS(Grid) but does not belong to Rec(Grid). In fact, the recognizable subsets of Grid are finite unions of products of subsets in $\text{Reg}((\mathbb{N}, +))$.

The following proposition is well-known in the case of AFS(G).

Proposition 7. *For all G, if $P \in \text{Rec}(G)$ and $Q \in \text{DAFS}(G)$ then $P \cap Q \in \text{DAFS}(G)$.*

4 Motivating Examples

4.1 The Grid

In this section, we consider the Cayley graph Grid (presented in Example 2) of the free commutative monoid with two generators. In [ES69], the authors establish that the regular subsets of this monoid are unambiguous. A regular subset R is unambiguous if it is accepted by a finite automaton A with a finite set of initial states and such that for all $r \in R$, there exists exactly one computation of A accepting r. Note that the run on the Cayley graph of the monoid of an unambiguous automaton is, in general, neither deterministic (as several initial states are allowed) nor complete (as the definition does not imply the existence of a path reaching the elements that do not belong to R). In fact, the unambiguous regular subsets are in general not closed under complementation.

The proof that Grid satisfies the strong Kleene equality is quite involved and starts from the unambiguous characterization of [ES69].

Theorem 1. *The Cayley graph* Grid *of the free commutative monoid with two generators satisfies the strong Kleene equality.*

We conjecture that this result extends to the Cayley graphs of freely generated monoids with an arbitrary number of generators.

4.2 Graphs of Deterministic Pushdown Automata

In [Sén96], the author proves that the regular subsets of a context-free group form a Boolean algebra. In [MS85], Muller and Schupp proved that Cayley graphs of context-free groups are rooted deterministic pushdown transition graphs. We establish that all rooted deterministic pushdown transition graphs satisfy the strong Kleene equality. In particular, it follows that the Cayley graphs of the context-free groups satisfy the strong Kleene equality.

Recall that a (real-time) *pushdown automaton* is a finite set R of rules of the form (pA, a, qU) with $p, q \in Q, A \in P, U \in P^*, a \in \Sigma$, where Q, P, Σ are disjoint alphabets of respectively *states*, *pushdown letters* and *labels*. A configuration of the pushdown graph is a word qw where $q \in Q$ and $w \in P^*$. The *transition graph* $P(R)$ of any pushdown automaton R is the uncolored graph $P(R) := \{(uw, a, vw) \mid (u, a, v) \in R \wedge w \in P^*\}$. The *rooted transition graph* $P(R, r)$ of R from any configuration $r \in QP^*$ is the graph $P(R, r)$ obtained from $P(R)$ by coloring the vertex r by a color ι and by restricting to the vertices accessible from r. Figure 6 illustrates this notion.

Theorem 2. *Every rooted deterministic pushdown transition graph G satisfies the strong Kleene equality.*

This result extends to the rooted deterministic prefix-recognizable graphs introduced in [Cau96]. A prefix-recognizable relation on words over a finite alphabet Γ is a finite union of relations of the form $(U \times V) \cdot W$ where U,V and W

(pA, a, PAA) $Q = \{p, q\}$
(pA, b, q) $P = \{A, B\}$
(qA, c, q)

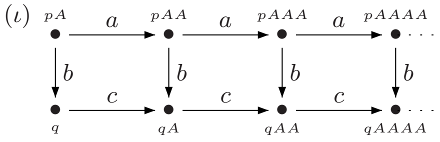


Fig. 6. A pushdown transition graph rooted in pA

belong to $\mathrm{Reg}(\Gamma^*)$. A prefix-recognizable graph labeled by Σ is an uncolored graph defined by a family $(R_a)_{a\in\Sigma}$ of prefix-recognizable relations on words over Γ^* and is equal to $\{\,(u, a, v) \mid u, v \in \Gamma^*, a \in \Sigma \text{ and } (u, v) \in R_a\,\}$. For any prefix-recognizable graph G, the prefix-recognizable graph $G_{/r}$ rooted in $r \in V_G$ is the graph obtained by restricting G to the set of vertices accessible from r and adding the color ι on r. Figure 7 presents a rooted deterministic prefix-recognizable graph which is not a rooted pushdown graph.

$\Sigma = \{a, b\}$
$\Gamma = \{c\}$
$R_a = (\{\varepsilon\}, \{c\}) \cdot c^*$
$R_b = (c^+, \{\varepsilon\}) \cdot c^*$

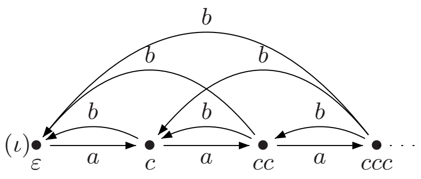


Fig. 7. A prefix-recognizable graph rooted in ε

Theorem 3. *Every rooted deterministic prefix-recognizable graph satisfies the strong Kleene equality.*

4.3 Free Product of Rooted Graphs

In order to obtain more graphs satisfying the Kleene equality, we consider the free product of rooted graphs. In [Sak87], the author established that for two disjoint monoids $\mathcal{M}$ and $\mathcal{N}$ such that $\mathrm{Reg}(\mathcal{M})$ and $\mathrm{Reg}(\mathcal{N})$ are Boolean algebras then the regular subsets of the free product of $\mathcal{M}$ and $\mathcal{N}$ form a Boolean algebra. We naturally extend the free product of two monoids to rooted graphs and show that on Cayley graphs, the free product preserves the strong Kleene equality.

Let G and H be two rooted graphs with respective roots r_G colored by g and r_H colored by h such that $V_G \cap V_H = \Sigma_G \cap \Sigma_H = \emptyset$ and $C_G = \{g\}$ and $C_H = \{h\}$. We take $V'_G = V_G \setminus \{r_G\}$ and $V'_H = V_H \setminus \{r_H\}$.

The set of vertices of the free product $G \otimes H$ of G and H is $S_G \cup S_H$ where $S_G = (V'_G V'_H)^* V'_G \cup (V'_H V'_G)^*$ and $S_H = (V'_H V'_G)^* V'_H \cup (V'_G V'_H)^*$. The graph $G \otimes H$ is rooted at empty sequence ε and defined by:

$$G \otimes H := \bigcup_{u \in S_G} u \cdot [H] \;\cup\; \bigcup_{u \in S_H} u \cdot [G] \;\cup\; \{(\iota, \varepsilon)\}$$

where $[G]$ (resp. $[H]$) designates the graph obtained by renaming the root r_G (resp. r_H) by the empty sequence and for all graph $u \cdot [G]$ (resp.$u \cdot [H]$) is the graph $\{ (uv, a, uw) \mid (v, a, w) \in G \}$ (resp. $\{ (uv, a, uw) \mid (v, a, w) \in H \}$).

In particular, the free product of the Cayley graphs of two disjoint monoids is the Cayley graph of the free product of these two monoids.

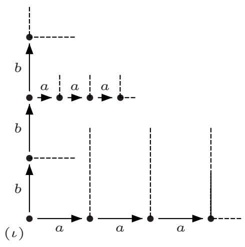


Fig. 8. The free product of two semi-lines labeled by a and by b respectively

A first simple case in which the free product preserves the strong Kleene equality is when one of the two graphs has no incoming edge to its root.

Proposition 8. *If G has no incoming edge to its root and if G and H both satisfy the Kleene equality then their free product $G \otimes H$ also satisfies the Kleene equality.*

In particular, it follows that the free-product of Grid and of any deterministic rooted pushdown graph satisfy the strong Kleene equality.

Theorem 4. *For any two disjoint monoids $\mathcal{M}$ and $\mathcal{N}$ finitely generated by $P \subset M$ and $Q \subset N$, if the $\mathcal{C}(\mathcal{M}, P)$ and $\mathcal{C}(\mathcal{N}, Q)$ both satisfy the strong Kleene equality then $\mathcal{C}(\mathcal{M}, P) \otimes \mathcal{C}(\mathcal{N}, Q)$ also satisfies the strong Kleene equality.*

The proof of Theorem 4 is an adaptation of Theorem 5.2 of [Sak87].

Example 5. Consider the free partially commutative monoid with four generators a, b, c, d satisfying the equations $ab = ba$ and $cd = dc$. Its Cayley graph is the free product of two grids respectively labeled by $\{a, b\}$ and $\{c, d\}$. By Theorem 4 and Theorem 1, its Cayley graph satisfies the strong Kleene equality.

5 Conclusion

In this article, we introduced the natural notion of deterministic acceptance by finite automata on colored graphs. We showed that it allows one to extend the Kleene theorem on free monoid to richer algebraic structures such as the free commutative monoid with two generators, the context-free groups and the trace monoid with a transitive independence relations with at most two independent generators (as their Cayley graphs are free products of grids and lines). We think that this notion brings new insight on the closure by complementation of the regular subsets of these monoids.

This work leaves several open questions. In particular, we conjecture that infinite grids of arbitrary dimension satisfy the strong Kleene equality. It remains to extend the notion of deterministic acceptance to capture the equational subsets of richer structures such as for example the canonical graphs associated to stacks of stacks [Car05]. Finally we can readily extend these notions to relational structures. However it remains to exhibit pertinent relational structures for which the equational subsets form a Boolean algebra.

References

[Büc64] J. Büchi. Regular canonical systems. *Arch. Math. Logik Grundlag.*, 6:91–111, 1964.

[Car05] A. Carayol. Regular sets of higher-order pushdown stacks. In *Proc. MFCS '05*, pages 168–179, 2005.

[Cau96] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proc. ICALP 96*, volume 1099 of *LNCS*, pages 194–205, 1996.

[Cay78] A. Cayley. On the theory of groups. *Proc. London Math. Soc.*, 9:126–133, 1878.

[Cou89] B. Courcelle. On recognizable sets and tree automata. In *Resolution of Equations in Algebraic Structures*. Academic Press, 1989.

[ES69] S. Eilenberg and M. Schützenberger. Rational sets in commutative monoids. *J. Algebra*, 13:344–353, 1969.

[GS64] S. Ginsburg and E. Spanier. Bounded algol-like languages. *Trans. Amer. Math. Soc.*, 113:333–368, 1964.

[MS85] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *TCS*, 37:51–75, 1985.

[MW67] J. Mezei and J. Wright. Algebraic automata and context free sets. *Information and Control*, 11:3–29, 1967.

[Sak87] J. Sakarovich. On regular trace languages. *TCS*, 52:59–75, 1987.

[Sén96] G. Sénizergues. On the rational subsets of the free group. *Acta Informatica*, 33:281–296, 1996.

On the Repetition Threshold for Large Alphabets

Arturo Carpi

Dipartimento di Matematica e Informatica, Università di Perugia, Italy
carpi@dipmat.unipg.it

Abstract. The (maximal) exponent of a finite non-empty word is the ratio among its length and its period. Dejean (1972) conjectured that for any $n \geq 5$ there exists an infinite word over n letters with no factor of exponent larger than $n/(n-1)$. We prove that this conjecture is true for $n \geq 38$.

1 Introduction

The existence of infinite words on a finite alphabet without adjacent repeats of a same factor is one of the oldest results in Combinatorics on Words [1,14,15] (see also [2]). A stronger request was considered by Dejean [6].

We recall that the (maximal) *exponent* of a non-empty finite word is the ratio among its length and its period. The supremum of the exponents of the factors of an infinite word is usually called the *critical exponent*. For instance, the critical exponent of Thue-Morse word is 2 [14] and that of Fibonacci word is $(5+\sqrt{5})/2$ [9]. For any $n \geq 2$, the minimal critical exponent of infinite words on n letters is called the *repetition threshold on n letters* [3]. Thus, as any binary word of length 4 has a factor of exponent 2, the repetition threshold on 2 letters is 2.

Dejean [6] proved that the repetition threshold on 3 letters is 7/4. Dejean has also showed that for $n \geq 5$, the repetition threshold on n letters is not smaller than $n/(n-1)$ while if $n = 4$, then it is not smaller than 7/5. She conjectured that these are the actual values of the repetition threshold. This conjecture has been proved to be true for $n = 4$ by Pansiot [12] and, with extensive use of a computer, for $5 \leq n \leq 11$ by Moulin-Ollagnier [11] and, more recently, for $12 \leq n \leq 14$ by Mohammad-Noori and Currie [10].

The goal of this paper is the proof of Dejean's conjecture for alphabet with at least 38 letters.

We mention that a generalization of the repetition threshold, taking into account not only the exponent but also the length of the factors has been considered in [7]. Binary or ternary words avoiding certain fractional powers are studied in [5,8,13].

Pansiot [12] showed that any infinite word on n letters of critical exponent $n/(n-1)$ can be obtained as the result of a sequential transduction of a binary word. As noticed in [11] the sequential transducer can be identified with a suitable morphism φ_n of the two-generated free monoid onto the symmetric group on n objects. In Section 3 we give a condition on the factors of an infinite binary word w ensuring that w is transduced into a word on n letters of critical exponent $n/(n-1)$.

In our analysis a central role is played by binary words which are mapped by φ_n into the stabilizer of k suitable points (k-stabilizing words of order n). Indeed, it turns out that if an infinite binary word contains a factor which is a 'too short' k-stabilizing word, then Pansiot transducer maps it into a word on n letters with critical exponent larger than $n/(n-1)$.

In Section 4 we introduce, for any integer $n \geq 8$, a morphism f from a finite alphabet into a binary alphabet with the property that 'short' k-stabilizing words of order n do not occur as factors of the words in the image of f.

In Section 5 we give a condition on the factors of a word w ensuring that the factors of $f(w)$ satisfy the condition established in Section 3. Finally, in Section 6 we produce infinite words satisfying such a condition, for any $n \geq 38$. This allows us to conclude that for alphabet with 38 or more letters, Dejean's conjecture is true.

2 Preliminaries

Let A be a finite non-empty set, or *alphabet*, and A^* be the free monoid generated by A. The elements of A are usually called *letters* and those of A^* *words*. The identity element of A^* is called *empty word* and denoted by ε. We set $A^+ = A^* \setminus \{\varepsilon\}$.

A word $v \in A^+$ can be written uniquely as a sequence of letters as $v = a_1a_2 \cdots a_\ell$, with $a_i \in A$, $1 \leq i \leq \ell$, $\ell > 0$. The integer ℓ is called the *length* of v and denoted $|v|$. The length of ε is 0. For any $v \in A^*$ and $a \in A$, $|v|_a$ denotes the number of occurrences of the letter a in v.

Let $v \in A^*$. The word u is a *factor* of v if there exist words r, s such that $v = rus$. A factor u of v is called *proper* if $u \neq v$. If $v = us$, for some word s (resp., $v = ru$, for some word r), then u is called a *prefix* (resp., a *suffix*) of v. For any $v \in A^*$, we denote by Fact(v), the set of its factors and respectively by Pref(v) and Suff(v) the sets of its proper prefixes and suffixes. For any $X \subseteq A^*$, we set

$$\mathrm{Fact}(X) = \bigcup_{v \in X} \mathrm{Fact}(v)\,.$$

An element of Fact(X) will be also called a *factor* of X.

Any non-empty word v can be uniquely factorized $v = u^k u'$ with $k \geq 1$, $u' \in \mathrm{Pref}(u)$ and $|u|$ minimal. The integer $p = |u|$ and the ratio $e = |v|/p$ are called respectively the (minimal) *period* and the (maximal) *exponent* of v.

An *infinite word* w on the alphabet A is any unending sequence $w = (a_i)_{i \geq 1}$ of letters of A. The set of all the infinite words over A is denoted by A^ω. A *factor* of w is any word $a_i a_{i+1} \cdots a_j$, with $1 \leq i \leq j$, as well as the empty word. Also in the case that w is an infinite word, the set of factors of w is denoted by Fact(w).

The *critical exponent* of an infinite word is the supremum of the exponents of its non-empty factors. It is possible to prove that if A is a n-letter alphabet, with $n \geq 2$, then the set of critical exponents of the words of A^ω has a minimum. According to [3], it is called the *repetition threshold* on n letters. As shown in [6], the repetition threshold on n letters is not smaller than $n/(n-1)$.

We say that a finite or infinite word w on the alphabet A *avoids* a set $X \subseteq A^*$ if $X \cap \mathrm{Fact}(w) = \emptyset$. A set of words Y is said to *avoid* X if all words of Y avoid X.

Let n be a positive integer. We shall denote by $\mathbb{S}_n$ the *symmetric group* on n objects. Thus, the elements of $\mathbb{S}_n$ are the *permutations* of the set $A_n = \{1, 2, \ldots, n\}$. For any $\alpha \in \mathbb{S}_n$ and $a \in A_n$ the image of a by α will be denoted by $a\alpha$. A permutation α is a *k-cycle* if there exist k distinct elements $a_1, \ldots, a_k \in A_n$, such that $a_i\alpha = a_{i+1}$, $1 \leq i \leq k-1$, $a_k = a_1$ and any other element of A_n is fixed by α. Such an α will be denoted by $\alpha = (a_1\, a_2\, \cdots\, a_k)$. As is well known, any permutation can be written as the product of pairwise disjoint cycles and its order is equal to the least common multiple of the lengths of the cycles.

3 Pansiot Transduction

Let $n \geq 2$ be an integer and $B = \{0, 1\}$. We consider the morphism $\varphi_n : B^* \to \mathbb{S}_n$ defined by

$$\varphi_n(0) = (1\ 2\ \cdots\ n-1)\,, \qquad \varphi_n(1) = (1\ 2\ \cdots\ n)\,.$$

Since for $1 \leq i \leq n-2$ one has $i\varphi_n(0) = i\varphi_n(1) = i+1$, one easily derives that if $u \in B^*$ and $i \leq n - |u| - 1$, then

$$i\varphi_n(u) = i + |u|\,. \tag{1}$$

Let $w = (b_i)_{i \geq 1}$ be an infinite word on the alphabet B. We shall denote by $\gamma_n(w)$ the infinite word $(a_i)_{i \geq 1}$ on the alphabet $A_n = \{1, 2, \ldots, n\}$ defined by

$$a_i = 1(\varphi_n(b_1 b_2 \cdots b_i))^{-1}\,, \quad i \geq 1 \tag{2}$$

Thus, we have defined a map $\gamma_n : B^\omega \to A_n^\omega$. The map γ_n was introduced in [12] as the output of a sequential transducer. It was proved that any infinite word on n letters whose critical exponent is smaller than $(n-1)/(n-2)$ can be obtained by renaming the letters of a word of $\gamma_n(B^\omega)$ (cf. [12, Lemme 2.3]).

Let $1 \leq k < n$. We say that a word $u \in B^+$ is a *k-stabilizing* word (*of order* n) if $\varphi_n(u)$ fixes the points $1, 2, \ldots, k$. We shall denote by $\mathrm{Stab}_n(k)$ the set of k-stabilizing words of order n. Next lemma establishes a correspondence among the occurrences of k-stabilizing words in an infinite binary word w and the 'repetitions' of a same factor of length k in $\gamma_n(w)$.

Lemma 1. *Let $w = (b_i)_{i \geq 1}$ be an infinite word on the alphabet B and set $\gamma_n(w) = (a_i)_{i \geq 1}$. For $1 \leq i < j$, $1 \leq k \leq n-1$, one has*

$$a_i a_{i+1} \cdots a_{i+k-1} = a_j a_{j+1} \cdots a_{j+k-1}$$

if and only if $b_{i+k} b_{i+k+1} \cdots b_{j+k-1}$ is k-stabilizing.

Proof. For $0 \le s \le k-1$, in view of (1) and (2) one has

$$\begin{aligned} a_{j+s}\varphi_n(b_1\cdots b_{j+k-1}) &= 1(\varphi_n(b_1\cdots b_{j+s}))^{-1}\varphi_n(b_1\cdots b_{j+k-1}) \\ &= 1\varphi_n(b_{j+s+1}\cdots b_{j+k-1}) = k-s\,, \\ a_{i+s}\varphi_n(b_1\cdots b_{j+k-1}) &= 1(\varphi_n(b_1\cdots b_{i+s}))^{-1}\varphi_n(b_1\cdots b_{j+k-1}) \\ &= 1\varphi_n(b_{i+s+1}\cdots b_{i+k-1})\varphi_n(b_{i+k}\cdots b_{j+k-1}) \\ &= (k-s)\varphi_n(b_{i+k}\cdots b_{j+k-1})\,. \end{aligned}$$

Thus one has $a_{i+s} = a_{j+s}$ if and only if $k-s$ is fixed by the permutation $\varphi_n(b_{i+k}\cdots b_{j+k-1})$. The conclusion follows. □

A *kernel repetition* (of *order* n) is any word of the form uu' with $u \in \ker\varphi_n$, $u' \in \mathrm{Pref}(u)$ and

$$|u| < (|u'| + n - 1)(n-1)\,.$$

In [11] it is noticed that if w avoids kernel repetitions then $\gamma_n(w)$ avoids all 'sufficiently long' factors of exponent larger than $n/(n-1)$. We can use the previous lemma to give a stronger condition ensuring that $\gamma_n(w)$ has critical exponent $n/(n-1)$.

Proposition 1. *Let w be an infinite word on the alphabet B and $n \ge 5$ be an integer. If no factor of w is a kernel repetition and for all $k < n$ no factor of w of length smaller than $k(n-1)$ is k-stabilizing, then the critical exponent of $\gamma_n(w)$ is $n/(n-1)$.*

Proof. Set $w = (b_i)_{i\ge 1}$ and $\gamma_n(w) = (a_i)_{i\ge1}$. By contradiction, suppose that $\gamma_n(w)$ has a factor r of exponent $e > n/(n-1)$. We may assume with no loss of generality that $e \le 2$. Thus $r = vv'$, with $v \in A_n^*$, v' a prefix of v and $|v'| = k > |v|/(n-1)$. Since $r = a_i a_{i+1}\cdots a_{i+|r|-1}$ for a suitable $i \ge 1$, one derives

$$v' = a_i a_{i+1}\cdots a_{i+k-1} = a_j a_{j+1}\cdots a_{j+k-1}\,,$$

where $j = i + |v|$, so that $k(n-1) > j - i$. If $k < n$ then by Lemma 1, $u = b_{i+k}b_{i+k+1}\cdots b_{j+k-1}$ is a k-stabilizing word of length $|u| = j-i < k(n-1)$, which is a contradiction. Thus we assume $k \ge n$. Then one has

$$a_{i+t}a_{i+t+1}\cdots a_{i+t+n-2} = a_{j+t}a_{j+t+1}\cdots a_{j+t+n-2}\,, \quad 0 \le t \le k-n+1\,.$$

Since $\mathrm{Stab}_n(n-1) = \ker\varphi_n$, by Lemma 1 one derives $b_{i+t+n-1}\cdots b_{j+t+n-2} \in \ker\varphi_n$. Hence, for $0 \le t \le k-n$ one has

$$\varphi_n(b_{i+t+n-1}\cdots b_{j+t+n-1}) = \varphi_n(b_{i+t+n-1}) = \varphi_n(b_{j+t+n-1})\,,$$

and consequently $b_{i+t+n-1} = b_{j+t+n-1}$. Set

$$u = b_{i+n-1}\cdots b_{j+n-2}\,, \quad u' = b_{i+n-1}\cdots b_{i+k-1} = b_{j+n-1}\cdots b_{j+k-1}\,.$$

One easily verifies that uu' is a kernel repetition and a factor of w. This is a contradiction. □

The previous proposition shows that in order to prove Dejean conjecture for a n-letter alphabet, it is sufficient to find an infinite binary word w satisfying the following conditions:

- for all $k < n$ no factor of w of length smaller than $k(n-1)$ is a k-stabilizing word of order n,
- no factor of w is a kernel repetition of order n.

4 A Binary Encoding

In the sequel we assume that $n \geq 8$ is a fixed integer. For this reason, we shall omit the index n in φ_n and $\mathrm{Stab}_n(k)$, whenever no confusion arises. In this section we introduce a uniform morphism from a finite alphabet into B^*. As we shall see in the sequel, at least when n is sufficiently large, the image of this morphism avoids, for all $k < n$, k-stabilizing words of length smaller than $k(n-1)$.

We set $p = \lfloor n/2 \rfloor$, $m = \lfloor (p-1)/3 \rfloor$. Moreover, we denote by $\widehat{y}$ the word defined by

$$\widehat{y} = \begin{cases} (01)^p & \text{if } n = 2p+1\,, \\ 1(01)^{p-1} & \text{if } n = 2p\,. \end{cases}$$

In both cases, $\widehat{y} = \widehat{x}\,(01)^{3m}$ for a suitable $\widehat{x} \in B^*$. We introduce the morphism $f : A_m^* \to B^*$ defined by

$$f(a) = \widehat{y}^{\,p}\,\widehat{x}\;(101101)^{m-a}\,010101\,(101101)^{a-1}\,, \quad a \in A_m\,.$$

We notice that for any $a \in A_m$,

$$|f(a)| = (n-1)(p+1)\,.$$

We set

$$\tau = \varphi(010101)\,, \qquad \rho = \varphi(\widehat{y}\,)\,, \qquad \sigma = \varphi(101)\,(\varphi(010))^{-1}\,.$$

It is not difficult to verify that ρ and σ are cycles and, more precisely,

$$\rho = (n \;\; 2p-1 \;\; 2p-3 \;\; 2p-5 \;\; \cdots \;\; 1)\,, \tag{3}$$

$$\sigma = (n-3 \;\; n-2 \;\; n \;\; n-1)\,.$$

For any $a \in A_m$ we set $\sigma_a = \tau^{-a}\sigma\tau^a$. One easily verifies that $\sigma_1 = (4\;3\;5\;6)$ and, more generally,

$$\sigma_a = (6a-2 \;\; 6a-3 \;\; 6a-1 \;\; 6a)\,, \quad a \in A_m\,. \tag{4}$$

Thus, the permutations σ_a are pairwise disjoint 4-cycles and therefore they commute.

Let us consider the morphism $\psi : A_m^* \to \mathbb{S}_n$ defined by $\psi(v) = \varphi(f(v))$, $v \in A_m^*$. The morphism ψ is described by the following lemma.

Lemma 2. *For any $a \in A_m$ one has*

$$\psi(a) = \prod_{c \in A_m \setminus \{a\}} \sigma_c \,.$$

Proof. One can easily prove, by induction on k, that for any $k \in A_m$, $(\sigma\tau)^k = \tau^k \prod_{c=1}^{k} \sigma_c$. Since, moreover, $\varphi(101101) = \varphi(101)\varphi(010)^{-1}\varphi(010101) = \sigma\tau$, one derives

$$\varphi((101101)^{m-a}) = \left(\tau^m \prod_{c=1}^{m} \sigma_c\right)\left(\tau^a \prod_{c=1}^{a} \sigma_c\right)^{-1} = \tau^m \left(\prod_{c=a+1}^{m} \sigma_c\right)\tau^{-a}$$

and

$$\varphi((101101)^{a-1}) = \tau^{a-1}\left(\prod_{c=1}^{a-1} \sigma_c\right),$$

so that

$$\varphi((101101)^{m-a}\,010101\,(101101)^{a-1}) = \tau^m \prod_{c \in A_m \setminus \{a\}} \sigma_c$$

and, therefore, $\psi(a) = \rho^p \varphi(\widehat{x}\,)\tau^m \prod_{c \in A_m \setminus \{a\}} \sigma_c$. Since $\rho = \varphi(\widehat{y}\,) = \varphi(\widehat{x}\,)\tau^m$, and by (3), the order of ρ is $p+1$, the conclusion follows. □

From previous lemma, one has that for all $v \in A_m^*$,

$$\psi(v) = \prod_{a \in A_m} \sigma_a^{|v| - |v|_a} \,. \tag{5}$$

For our purposes the main property of the morphism f is given by the following

Proposition 2. *Let $n \geq 28$. For any $k = 1, \ldots, n-1$, $f(A_{m-1}^*)$ avoids k-stabilizing words of length smaller than $k(n-1)$.*

The proof of this proposition is rather complex. It makes use of the following technical lemma concerning the action of φ on the factors of $f(A_m^*)$

Lemma 3. *Let $w \in A_m^*$ and $u \in \mathrm{Fact}(f(w))$. Then there exist $v_1, v_2 \in \mathrm{Fact}(w)$, $x_1, x_2, x_3, x_4 \in B^*$ and integers h_1, h_2, h_3, h_4 such that*

$$\varphi(u) = (\varphi(x_1))^{-1}\rho^{-h_1}\psi(v_1)\rho^{h_2}\varphi(x_2) = \varphi(x_3)\rho^{-h_3}\psi(v_2)\rho^{h_4}(\varphi(x_4))^{-1} \tag{6}$$

$$\begin{aligned} |u| &= (n-1)((p+1)|v_1| + h_2 - h_1) + |x_2| - |x_1| \\ &= (n-1)((p+1)|v_2| + h_4 - h_3) + |x_3| - |x_4| \,, \end{aligned} \tag{7}$$

$$0 \leq h_i \leq p\,,\ 0 \leq |x_i| \leq n-2\,,\ i = 1,2,3,4\,, \quad |x_1x_3|, |x_2x_4| \leq n-1\,. \tag{8}$$

Moreover, if $h_1 < p$, then $x_3 \in \mathrm{Suff}(\widehat{y}\,)$, if $h_2 < p$, then $x_4 \in \mathrm{Suff}(\widehat{y}\,)$, if $h_1 = h_2 = p$, $x_1 \neq \varepsilon$ and $x_2 \neq \varepsilon$ then $v_1c = c'v_2$ for suitable $c, c' \in A_m$.

For the sake of brevity, we limit ourselves to outline the main steps of the proof of Proposition 2. The complete proof is in [4].

First, we remark that no 1-stabilizing word u of length $|u| < n-1$ exists. Indeed, if $0 < |u| < n-1$, then by (1) one has $1\varphi(u) = 1 + |u| \neq 1$. A more complex combinatorial analysis shows that a 2-stabilizing word u of length $|u| < 2(n-1)$ has necessarily the form $u = u'00$ or $u = u'111$ with $|u'| = n-3$. Since 00 and 111 are not factors of $f(A_m^*)$, we conclude that such an u cannot occur in $f(A_m^*)$. It is also possible to prove that a 3-stabilizing word u of length $|u| < 3(n-1)$ either contains one of the factors 00 or 111 or has the form $u = y'xyx$ with $|x| = 4$ and $|y| = |y'| = n-4$. However, it turns out that $f(A_m^*)$ has no factor of this form.

In the next step, one shows that if u is a 4-stabilizing factor of $f(A_m^*)$, then $|u| \geq (n-1)(p+1)$. The proof is based on Lemma 3. Finally, one verifies that if $n > 16$ and u is a 16-stabilizing factor of $f(A_{m-1}^*)$, then $|u| > 3(n-1)(p+1)$.

Now suppose $n \geq 28$ and let $u \in \mathrm{Fact}(f(A_{m-1}^*)) \cap \mathrm{Stab}(k)$. As we have seen, if $k \leq 3$, then $|u| \geq k(n-1)$ and if $4 \leq k \leq p+1$, then $|u| \geq (n-1)(p+1) \geq k(n-1)$. Finally, if $p+2 \leq k \leq n-1$, then since $p \geq 14$, one has $k \geq 16$ so that $|u| > 3(n-1)(p+1) > k(n-1)$.

5 Kernel Repetitions

We call a ψ-*kernel repetition* (of order n) any word $s \in A_m^*$ of the form vv' with

$$v \in \ker\psi\,, \quad v' \in \mathrm{Pref}(v)\,, \quad |v| < (|v'|+3)(n-1)\,.$$

In this section we shall prove that if $w \in A_{m-1}^*$ avoids ψ-kernel repetitions, then $f(w)$ avoids kernel repetitions.

Proposition 3. *Let $w \in A_{m-1}^*$ and $u \in \mathrm{Fact}(f(w)) \cap \ker\varphi$. Then there exists $v \in \mathrm{Fact}(w) \cap \ker\psi$ such that $|u| = |f(v)|$.*

Proof. Let $v_1, v_2 \in \mathrm{Fact}(w)$, $x_1, x_2, x_3, x_4 \in B^*$, $h_1, h_2, h_3, h_4 \geq 0$ satisfy the statement of Lemma 3.

As $u \in \ker\varphi$, from (6) one has

$$\psi(v_1) = \rho^{h_1}\varphi(x_1)(\varphi(x_2))^{-1}\rho^{-h_2}\,, \quad \psi(v_2) = \rho^{h_3}(\varphi(x_3))^{-1}\varphi(x_4)\rho^{-h_4}\,. \tag{9}$$

First, we verify that $|x_1| = |x_2|$. Since by (3), (4) and (5), 2 is fixed by ρ and $\psi(v_1)$, one derives from (9) that $2\varphi(x_1) = 2\varphi(x_2)$. If $|x_1|, |x_2| \leq n-3$, then by (1) one obtains $2\varphi(x_i) = 2 + |x_i|$, $i = 1, 2$ so that $|x_1| = |x_2|$. Thus assume, for instance, $|x_1| = n-2 > |x_2|$ and write $x_1 = x'b$, with $b \in B$. Again by (1) one obtains $2\varphi(x') = n-1$ so that $2\varphi(x_1) = (n-1)\varphi(b) \in \{1, n\}$ and $2\varphi(x_2) = 2 + |x_2| \notin \{1, n\}$ which is a contradiction. We conclude that $|x_1| = |x_2|$. From (7) and (8) one derives also $|x_3| = |x_4|$.

We distinguish three cases, according to the values of h_1 and h_2.

Case 1: $h_1 = p$.
In this case, $n\rho^{h_1} = 1$. Since $1\varphi(x_1) = 1\varphi(x_2) = 1 + |x_1|$ and n is fixed by $\psi(v_1)$, by (9) one has $n = 1\rho^{-h_2} = n\rho^{-(h_2+1)}$. This equation implies that $h_2 = p$, so

that by (7), $|u| = |f(v_1)|$. If $x_1 = x_2 = \varepsilon$, one derives that $\psi(v_1)$ is the identity, proving the statement. Thus we assume $x_1, x_2 \neq \varepsilon$, so that, from Lemma 3,

$$v_1 c = c' v_2 \,,$$

for suitable $c, c' \in A_m$. Now we verify that for any $a \in A_m$, either $|v_1|_a \equiv |v_1|$ or $|v_2|_a \equiv |v_2| \pmod 4$. Indeed, if $6a > |x_3|$, then $(6a - |x_4|)\varphi(x_3) = (6a - |x_4|)\varphi(x_4) = 6a$, so that $6a$ is fixed by $(\varphi(x_3))^{-1}\varphi(x_4)$. Since $6a$ is fixed also by ρ, from (9) one obtains that $6a$ is fixed by $\psi(v_2)$. However, from (5) one has $(6a)\psi(v_2) = (6a)\sigma_a^{|v_2| - |v_2|_a}$. Since σ_a is a 4-cycle moving $6a$, one derives $|v_2| \equiv |v_2|_a \pmod 4$. If, on the contrary, $6a \leq |x_3|$, then $6a + |x_1| \leq |x_1 x_3| \leq n - 1$ and therefore $(6a)\varphi(x_1) = (6a)\varphi(x_2) = 6a + |x_1|$. This implies that $6a$ is fixed by $\varphi(x_1)(\varphi(x_2))^{-1}$ and consequently by $\psi(v_1)$. Since $(6a)\psi(v_1) = (6a)\sigma_a^{|v_1| - |v_1|_a}$, one derives $|v_1| \equiv |v_1|_a \pmod 4$. Thus, for any $a \in A_m$, either

$$|v_1|_a \equiv |v_1| \quad \text{or} \quad |v_2|_a \equiv |v_2| \pmod 4 \,.$$

Since $w \in A_{m-1}^*$, one has $|v_1|_m = |v_2|_m = 0$ and therefore $|v_1| = |v_2| \equiv 0 \pmod 4$. Let $i \in \{1, 2\}$ be such that $|v_i|_c \equiv 0 \pmod 4$. For any $a \in A_m \setminus \{c, c'\}$ one has $|v_1|_a = |v_2|_a$, so that $|v_i|_a \equiv 0 \pmod 4$. Moreover, $|v_i|_{c'} = |v_i| - \sum_{a \in A_m \setminus \{c'\}} |v_i|_a \equiv 0 \pmod 4$. By (5) one derives that $v_i \in \ker \psi$. The conclusion follows.

Case 2: $h_2 = p$.
From (9) one has $(\psi(v_1))^{-1} = \rho^{h_2}\varphi(x_2)(\varphi(x_1))^{-1}\rho^{-h_1}$. If $h_2 = p$, one has $n\rho^{h_2} = 1$. Since n is fixed by $\psi(v_1)$ and 1 is fixed by $\varphi(x_2)(\varphi(x_1))^{-1}$, one derives $n = 1\rho^{-h_1} = n\rho^{-(h_1+1)}$. This equation implies that $h_1 = p$ so that we are reduced to Case 1.

Case 3: $h_1, h_2 < p$.
In this case, by Lemma 3 one has $x_3, x_4 \in \mathrm{Suff}(\widehat{y})$. Since $|x_3| = |x_4|$ one derives $x_3 = x_4$ so that from (9), $\psi(v_2) = \rho^{h_3 - h_4}$. Since $\psi(v_2)$ fixes 1, one derives $h_3 - h_4 = 0$. Thus $v_2 \in \ker \psi$ and, in view of (7), $|u| = |f(v_2)|$. □

Proposition 4. *Let $w \in A_{m-1}^*$. If a factor of $f(w)$ is a kernel repetition, then there is a ψ-kernel repetition occurring in w.*

Proof. We denote $K = (p+1)(n-1)$. Let r be a kernel repetition occurring in $f(w)$ and write $r = uu'$, with $u \in \ker \varphi$, $u' \in \mathrm{Pref}(u)$, $|u| < (|u'| + n - 1)(n - 1)$.

First, we consider the case that $|u'| \leq 2K$. In such a case one easily obtains $|u| < 3K(n-1)$. By Proposition 3 there exists $v \in \mathrm{Fact}(w) \cap \ker \psi$ such that $|u| = |f(v)| = K|v|$. Thus, $|v| < 3(n-1)$ so that taking $v' = \varepsilon$, $v = vv'$ is a ψ-kernel repetition.

Now suppose $|u'| > 2K$. By Proposition 3, one has $|u| = K\ell$, $\ell \geq 1$. Since uu' is a factor of $f(w)$ one can factorize

$$uu' = \xi f(v_0)\eta \,.$$

with $v_0 \in \mathrm{Fact}(w)$, $\xi, \eta \in B^*$, $|\xi|, |\eta| < K$. Moreover, $(\ell+2)K < |uu'| = K|v_0| + |\xi\eta| < K|v_0| + 2K$ so that $|v_0| > \ell$. Thus we can write $v_0 = vv'$, with $|v| = \ell$. Thus,

$$\xi f(v) = u\xi' , \quad u' = \xi' f(v')\eta , \quad |\xi| = |\xi'| .$$

Now, $\xi' f(v')$ is a prefix of u', u' is a proper prefix of u and u is a prefix of $\xi f(v)$. Hence, $\xi' f(v')$ is a proper prefix of $\xi f(v)$. Since, moreover, $|\xi| = |\xi'|$, we derive that $\xi = \xi'$ and $v' \in \mathrm{Pref}(v)$. From the equality $\xi f(v) = u\xi$, one easily derives that $v \in \ker\psi$. Since $|u'| = |\xi' f(v')\eta| < K(|v'|+2)$ one has

$$K|v| = |u| < (|u'| + n - 1)(n-1) < K(|v'|+3)(n-1)$$

so that $|v| < (|v'|+3)(n-1)$ and therefore vv' is a ψ-kernel repetition. □

6 Avoiding ψ-Kernel Repetitions

By the results of the previous sections, at least in the case $n \geq 28$, to construct an infinite word on n letters with critical exponent $n/(n-1)$, it is sufficient to find an infinite word on the alphabet A_{m-1} avoiding ψ-kernel repetitions. In this section we shall construct such a word for any $n \geq 38$.

Lemma 4. *One has*

$$A_{m-1}^* \cap \ker\psi = \{v \in A_{m-1}^* \mid \forall a \in A_{m-1} ,\ 4 \text{ divides } |v|_a\} .$$

Proof. Suppose $v \in A_{m-1}^* \cap \ker\psi$. By (5) one obtains that for all $a \in A_m$, 4 divides $|v| - |v|_a$. Since $|v|_m = 0$, one derives that 4 divides $|v|$ and, consequently, 4 divides $|v|_a$ for all $a \in A_{m-1}$.

Conversely, if $v \in A_{m-1}^*$ and 4 divides $|v|_a$ for all $a \in A_{m-1}$, then 4 divides also $|v| = \sum_{i=1}^{m-1} |v|_a$ and therefore $|v| - |v|_a$, for all $a \in A_m$. From (5) one derives $v \in \ker\psi$. □

Lemma 5. *Let $w_1 = (b_i)_{i\geq 1}$ be the infinite word on the alphabet $A_{m-1} \setminus A_{m-3}$ defined as follows:*

$$b_i = \begin{cases} m-1 & \text{if } i \equiv 1 \pmod 3 ,\\ m-2 & \text{if } i \equiv 2 \pmod 3 ,\\ b_{i/3} & \text{if } i \equiv 0 \pmod 3 ,\end{cases} \qquad i \geq 1 .$$

If one has $v \in A_m^*$, $v' \in \mathrm{Pref}(v) \cup \{v\}$, $vv' \in \mathrm{Fact}(w_1)$, and $|v'| \geq 3^k$, $k \geq 0$, then 3^k divides $|v|$.

Proof. Let $vv' = b_i b_{i+1} \cdots b_{i+|vv'|-1}$, $i \geq 1$. Then one has

$$b_i b_{i+1} \cdots b_{i+3^k-1} = b_{i+|v|} b_{i+|v|+1} \cdots b_{i+|v|+3^k-1} . \tag{10}$$

Set $|v| = 3^q t$, where 3^q is the maximal power of 3 dividing $|v|$ and assume, by contradiction, $q < k$. There exists $j \equiv |v| \pmod{3^k}$ such that $i \leq j \leq i+3^k-1$. One has $j = 3^q(t + 3^{k-q}h)$ and $j + |v| = 3^q(2t + 3^{k-q}h)$ for some integer h. Since 3 does not divide t, by the definition of w_1 one derives $b_j = b_t$, $b_{j+|v|} = b_{2t}$ and $b_t \neq b_{2t}$. This yields a contradiction because by (10), $b_j = b_{j+|v|}$. □

Lemma 6. *Let $w_2 = (c_i)_{i\geq 1}$ be the infinite word on the alphabet A_{m-3} defined as follows:*

$$c_i = \max\{a \in A_{m-3} \mid 4^{a-1} \text{ divides } i\}\,, \qquad i \geq 1\,.$$

For any $v \in \mathrm{Fact}(w_2) \cap \ker\psi$, 4^{m-3} divides $|v|$.

Proof. Set $|v| = 4^q t$, where 4^q is the maximal power of 4 dividing $|v|$ and assume, by contradiction, $q < m-3$. One has $v = c_i c_{i+1} \cdots c_{i+4^q t-1}$ for some $i \geq 1$. By definition for any $j \geq 1$, one has $c_j > q$ if and only if 4^q divides j. Thus

$$\sum_{a=q+1}^{m-3} |v|_a = \mathrm{Card}\{j \mid i \leq j \leq i + 4^q t - 1\,,\ 4^q \text{ divides } j\} = t\,.$$

Since $v \in \ker\psi$, by Lemma 4 one derives that 4 divides t. This contradicts the maximality of q. □

Proposition 5. *Let $w_1 = (b_i)_{i\geq 1}$ and $w_2 = (c_i)_{i\geq 1}$ be the infinite words considered in Lemmas 5 and 6. If $n \geq 44$, then the infinite word*

$$w = b_1 c_1 b_2 c_2 \cdots b_i c_i \cdots$$

avoids ψ-kernel repetitions.

Proof. By contradiction suppose that a ψ-kernel repetition r occurs in w. One has

$$r = vv'\,, \quad v \in \ker\psi\,, \quad v' \in \mathrm{Pref}(v)\,, \quad |v| \leq (|v'|+3)(n-1)\,.$$

Since $v \in A_{m-1}^* \cap \ker\psi$, by Lemma 4 one has that for all $a \in A_m$, 4 divides $|v|_a$. In particular, $|v|$ is even. Thus, deleting in v all the occurrences of the letters $m-2$ and $m-1$, one obtains a factor v_2 of w_2 such that $v_2 \in \ker\psi$ and $|v_2| = |v|/2$. By Lemma 6 one derives that 4^{m-3} divides $|v_2|$ and therefore $2 \cdot 4^{m-3}$ divides $|v|$.

Suppose $|v'| \geq 2$ and let k be the integer such that $2 \cdot 3^k \leq |v'| < 2 \cdot 3^{k+1}$. Deleting in r all the occurrences of the letters of A_{m-3}, one obtains a factor of r_1 of w_1 of the form $r_1 = v_1 v_1'$ where v_1' is a prefix of v_1, $|v_1| = |v|/2$ and $3^k \leq |v_1'| \leq 3^{k+1}$. By Lemma 5 one derives that 3^k divides $|v_1|$ and consequently $|v|$. Thus,

$$|v| \geq 2 \cdot 4^{m-3} 3^k > \frac{4^{m-3}}{3}|v'|\,.$$

We recall that $m = \lfloor (n-2)/6 \rfloor$, so that

$$n - 1 \leq 6(m+1)\,.$$

Since $n \geq 44$, one has $m \geq 7$ and therefore, as one easily verifies, $4^{m-3} \geq 32(m+1) \geq 16(n-1)/3$. Thus, one derives $|v| \geq 16(n-1)|v'|/9$. Since $|v| \leq (|v'|+3)(n-1)$ one obtains

$$|v'| + 3 \geq \frac{16}{9}|v'|$$

which implies $|v'| \leq 3$. Since $2 \cdot 4^{m-3}$ divides $|v|$, one obtains

$$|v| \geq 2 \cdot 4^{m-3} \geq \frac{32}{3}(n-1) > (|v'|+3)(n-1)\,,$$

which is a contradiction. □

Proposition 6. *Suppose* $38 \leq n \leq 43$. *Let* $w_1 = (b_i)_{i \geq 1}$ *and* $w_2 = (c_i)_{i \geq 1}$ *be the infinite words considered in Lemmas 5 and 6 and* $w = (d_i)_{i \geq 1}$ *be the infinite word defined by*

$$\begin{aligned} d_{2i} &= b_i\,,\ i \geq 1\,,\\ d_{4i+1} &= c_i\,,\ i \geq 0\,,\\ d_{4i+3} &= \begin{cases} 4 & \textit{if } (4i+3) \bmod (2 \cdot 4^4) < 4^4\,,\\ 5 & \textit{else,} \end{cases} \qquad i \geq 0\,. \end{aligned}$$

Then w *avoids* ψ*-kernel repetitions.*

Proof. Since $38 \leq n \leq 43$ one has $m = 6$. By contradiction suppose that a ψ-kernel repetition r occurs in w. One has

$$r = vv'\,,\quad v \in \ker\psi\,,\quad v' \in \mathrm{Pref}(v)\,,\quad |v| \leq (|v'|+3)(n-1)\,.$$

Since $v \in A_{m-1}^* \cap \ker\psi$, by Lemma 4 one has that for all $a \in A_m$, 4 divides $|v|_a$. In particular, 4 divides $|v|$. Thus, deleting in v all the occurrences of the letters $4 = m-2$ and $5 = m-1$, one obtains a factor v_2 of w_2 such that $|v| = 4|v_2|$ and $v_2 \in \ker\psi$. By Lemma 6 one derives that 4^3 divides $|v_2|$. Thus, $|v| = 4^4 h$ for some positive integer h and $(|v'|+3)(n-1) \geq 4^4 h$. As $n \leq 43$, from the latter inequality one derives $|v'| > 3$.

Since vv' is a factor of w and v' is a prefix of v, one has

$$v' = d_i d_{i+1} \cdots d_{i+k-1} = d_{i+4^4h} d_{i+4^4h+1} \cdots d_{i+4^4h+k-1}\,,$$

for some $i \geq 1$ and $k \geq 4$. There exists $j \equiv 3 \pmod 4$ such that $i \leq j \leq i+k-1$. By the previous equation, $d_j = d_{j+4^4h}$. By the definition of w, this implies that h is even. Thus, for $i' = \lceil i/2 \rceil$, $k' = \lfloor k/2 \rfloor$, $h' = h/2$ one has

$$b_{i'} b_{i'+1} \cdots b_{i'+k'-1} = d_{2i'} d_{2(i'+1)} \cdots d_{2i'+2k'-2}\,,$$

$$b_{i'+4^4h'} b_{i'+4^4h'+1} \cdots b_{i'+4^4h'+k'-1} = d_{2i'+4^4h} d_{2i'+4^4h+2} \cdots d_{2i'+4^4h+2k'-2}\,,$$

and therefore

$$b_{i'} b_{i'+1} \cdots b_{i'+k'-1} = b_{i'+4^4h'} b_{i'+4^4h'+1} \cdots b_{i'+4^4h'+k'-1}\,.$$

Let q be the integer such that $3^q \leq k' < 3^{q+1}$. By Lemma 5 one derives that 3^q divides h'. Thus,

$$|v| = 2 \cdot 4^4 h' \geq 2 \cdot 4^4 3^q > 42(k+3) \geq (|v'|+3)(n-1)$$

which is a contradiction. □

By Propositions 5 and 6, for any $n \geq 38$ there exists an infinite word $w = (d_i)_{i \geq 1}$ on the alphabet A_{m-1} avoiding ψ-kernel repetitions of order n. By Propositions 2 and 4 the infinite binary word

$$f(w) = f(d_1)f(d_2) \cdots f(d_i) \cdots$$

avoids kernel repetitions of order n and, for all $k < n$, k-stabilizing words of order n and length smaller than $k(n-1)$. By Proposition 1 we conclude that the critical exponent of $\gamma_n(f(w))$ is $n/(n-1)$. Since as showed in [6] the repetition threshold on n letters cannot be smaller than $n/(n-1)$, we have proved the following

Theorem 1. *For $n \geq 38$ the repetition threshold on n letters is $n/(n-1)$.*

References

1. J. Berstel, *Axel Thue's papers on repetition in words: a translation*, Publications du LaCIM, Université du Québec á Montréal **20** (1995).
2. J. Berstel, J. Karhumäki, Combinatorics on words: a tutorial, *Bulletin of the EATCS* **79** (2003) 178–228
3. F.-J. Brandenburg, Uniformly growing k-th power-free homomorphisms, *Theoret. Comput. Sci.* **23** (1983) 69–82
4. A. Carpi, On the repetition threshold for large alphabets, Dipartimento di Matematica e Informatica dell'Università di Perugia, Tech. rep. no. 5-2006 (2006)
5. M. Crochemore, P. Goralcik, Mutually avoiding ternary words of small exponent, *Int. J. Alg. Comp.* **1** (1991) 407–410
6. F. Dejean, Sur un théorème de Thue, *J. Combin. Th. A*, **13** (1972) 90–99
7. L. Ilie, P. Ochem, J. Shallit, A generalization of repetition threshold, in: J. Fiala *et al.* (eds.) *Proceedings MFCS 2004*, Lecture Notes in Comput. Sci., **3153**, Springer (Berlin, 2004) 818–826
8. J. Karhumäki, J. Shallit, Polynomial versus exponential growth in repetition-free binary words, *J. Comb. Theory Ser. A*, **105** (2004) 335–347
9. F. Mignosi, G. Pirillo, Repetitions in the Fibonacci infinite word, *RAIRO Inf. Theor. and Appl.* **26** (1992) 199–204
10. M. Mohammad-Noori, J. D. Currie, Dejean's conjecture and Sturmian words, *European J. Comb.*, to appear
11. J. Moulin-Ollagnier, Proof of Dejean's conjecture for alphabets with 5, 6, 7, 8, 9, 10 and 11 letters, *Theoret. Comput. Sci.* **95** (1992) 187–205
12. J.-J. Pansiot, A propos d'une conjecture de F. Dejean sur les répétitions dans les mots, *Discr. Appl. Math.* **7** (1984) 297–311
13. J. Shallit, Simultaneous avoidance of large squares and fractional powers in infinite binary words, *Int. J. Found. Comput. Sci.* **15** (2004) 317–327.
14. A. Thue, Uber unendliche Zeichenreihen, *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl.*, Christiania **7** (1906) 1–22 Reprinted in T. Nagell (ed.), *Selected Mathematical Papers of Axel Thue*, Universitetsforlaget, Oslo, 1977, pp. 139-158.
15. A. Thue, Uber die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. *Norske Vid. Selsk. Skr. Mat. Nat. Kl.*, Christiania **1** (1912), 1–67. Reprinted in T. Nagell (ed.), *Selected Mathematical Papers of Axel Thue*, Universitetsforlaget, Oslo, 1977, pp. 413-478.

Improved Parameterized Upper Bounds for Vertex Cover

Jianer Chen[1,⋆], Iyad A. Kanj[2,⋆⋆], and Ge Xia[3]

[1] Department of Computer Science, Texas A&M University, College Station, TX 77843, USA
chen@cs.tamu.edu

[2] School of Computer Science, Telecommunications and Information Systems, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604-2301, USA
ikanj@cs.depaul.edu

[3] Department of Computer Science, Lafayette College, Easton, PA 18042, USA
gexia@cs.lafayette.edu

Abstract. This paper presents an $O(1.2738^k + kn)$-time polynomial-space parameterized algorithm for Vertex Cover improving the previous $O(1.286^k + kn)$-time polynomial-space upper bound by Chen, Kanj, and Jia. The algorithm also improves the $O(1.2745^k k^4 + kn)$-time exponential-space upper bound for the problem by Chandran and Grandoni.

1 Introduction

This paper considers the parameterized Vertex Cover problem, abbreviated VC henceforth: given a graph G and a parameter k, decide if G has a vertex cover of at most k vertices. This problem was amongst the first few problems that were shown to be NP-hard [14]. In addition, the problem has been a central problem in the study of parameterized algorithms [11], and has applications in areas such as computational biochemistry and biology [6]. Since the development of the first parameterized algorithm for the problem by Sam Buss which runs in $O(kn + 2^k k^{2k+2})$ time [3], there has been an impressive list of improved algorithms for the problem [1,7,8,10,17,18,20]. The most recent algorithm for the problem running in polynomial space was presented in 1999 and gives the currently best time upper bound of $O(kn + 1.286^k)$ [7]. Algorithms using exponential space for the problem have also been proposed [5,7,18], amongst which the best runs in time $O(1.2745^k k^4 + kn)$ [5]. Most of the previous algorithms rely on exhaustive case-by-case analysis, and work under a conservative worst-case-scenario assumption. The analysis of these algorithms would consider the worst-case branch over numerous combinatorial cases, and derive an upper bound accordingly. In particular, the design phase of these algorithms (usually) did not provide the appropriate ground that the analysis phase could take advantage of

⋆ This author was supported in part by the NSF under grants CCF-0430683 and CCR-0311590.

⋆⋆ This author was supported in part by DePaul University Competitive Research Grant.

to derive better upper bounds than the ones claimed. Consequently, to improve the upper bounds, larger and larger sets of local structures had to be examined and processed differently. Examining these numerous structures and processing them differently on a case-by-case basis became very meticulous, rendering the verification and implementation of these algorithms very complicated and unpractical.

On the other hand, progress has been recently made on deriving computational lower bounds for the problem. It has been shown that unless all SNP problems are solvable in sub-exponential time, there is a constant $c_0 > 1$ such that Vertex Cover cannot be solved in time $c_0^k n^{O(1)}$ [4,15]. Therefore, from both the algorithmic and the complexity points of view, it becomes important to study how far we can push to lower the constant $c > 1$, such that the VC problem can be solved in time $c^k n^{O(1)}$.

In this paper we adopt a different approach to improve the time upper bound for the VC problem. Our goal was to design an algorithm that is simple and uniform, and that provides the tools and the ground for an insightful analysis of its running time. We came up with an algorithm that is very simple when compared to the (recent) previous algorithms. The algorithm keeps a list of prioritized "advantageous" structures at its disposal. At each stage it will pick the structure of highest priority (most advantageous structure). Picking such a structure can be easily done following few simple rules. When this structure is picked, the algorithm processes this structure very *uniformly*, and *obliviously*, in a way that is almost independent of what the structure is. As a matter of fact, there are *only* two different ways for processing *any* structure–that is, only two different branches–that the algorithm needs to distinguish. All the other operations performed by the algorithm are non-branching operations that process certain simple structures in the graph such as degree-1 and degree-2 vertices, and that set the stage for the subsequent branch performed by the algorithm to be efficient. The interleaving and ordering of these operations in the algorithm is very crucial, and is fully exploited by the analysis phase. The analysis phase however is lengthy, showing that regardless of the structure picked, the oblivious branching performed by the algorithm will yield the desired upper bound.

To be able to carry out all the above, a set of new techniques and generalization of some well-known and classical techniques have been introduced. A graph operation that is a generalizations of the *folding* operation [7], and a graph operation that is a specialization of the *struction* operation [12], have been developed. These operations help the algorithm remove several simple structures from the graph without the need to perform any branching. This makes analyzing the two branching operations performed in the resulting graph more insightful. The notion of a *tuple*, which was implicitly used by Robson [19], has been fully developed and exploited to prune the search space. Finally we perform a "local" amortized analysis to balance expensive branching operations by combining them with more efficient operations. Being able to perform this local amortized analysis is indebted to the careful interleaving and ordering of the operations in the algorithm, and not to the different way of processing each structure.

The presented algorithm runs in polynomial space, and has its running time bounded by $O(1.2738^k + kn)$. This is a significant improvement over the previous polynomial-space algorithm for the problem which runs in $O(1.286^k + kn)$ time. This also improves the exponential space $O(1.2745^k k^4 + kn)$-time algorithm by Chandran and Grandoni [5]. Most of the proofs in this paper are omitted due to lack of space.

2 Preliminaries and Structural Results

For a graph G we denote by $|G|$ the number of vertices in G. For a vertex v in G we denote by $N(v)$ the set of neighbors of v, $N[v]$ the set $N(v) \cup \{v\}$, and $d(v)$ the degree of v in G. For a set of vertices S in G, let $N(S)$ denote the set of neighbors of the vertices in S, and $N[S]$ the set $N(S) \cup S$. Let $\tau(G)$ denote the size of a minimum vertex cover of G. The following proposition from [7] is based on a theorem by Nemhauser and Trotter [16], usually referred to as the NT-theorem or the NT-decomposition.

Proposition 1 ([7]). *There is an algorithm of running time $O(kn + k^3)$ that, given an instance (G, k) of the* VC *problem where $|G| = n$, constructs another instance (G_1, k_1) of* VC *with $k_1 \leq k$ and $|G_1| \leq 2k_1$, such that $\tau(G) \leq k$ if and only if $\tau(G_1) \leq k_1$.*

We say that the instance (G_1, k_1) is the *kernel* of the instance (G, k). The NT-decomposition of (G, k) into (G_1, k_1) is said to be *non-trivial* if $|G_1| < |G|$. Proposition 1 allows us to assume, without loss of generality, that in an instance (G, k) of the VC problem the graph G contains at most $2k$ vertices.

For two vertices u and v we say that (u, v) is an *anti-edge* in G if (u, v) is not an edge in G. Let v_0 be a vertex in G with a set of neighbors $\{v_1, \cdots, v_p\}$. Construct a graph G' as follows: (1) remove the vertices $\{v_0, v_1, \cdots, v_p\}$ from G and introduce a new node v_{ij} for every anti-edge (v_i, v_j) in G where $0 < i < j \leq p$; (2) add an edge (v_{ir}, v_{js}) if $i = j$ and (v_r, v_s) is an edge in G; (3) if $i \neq j$ add an edge (v_{ir}, v_{js}); and (4) for every $u \notin \{v_0, \cdots, v_p\}$, add the edge (v_{ij}, u) if (v_i, u) or (v_j, u) is an edge in G. This completes the construction of G'. We say that the graph G' is obtained from G by applying the *struction* operation to the vertex v_0 in G [12] (see Figure 1 for an illustration).

Lemma 1. *Let v_0 be a vertex in G with a set of neighbors $\{v_1, \cdots, v_p\}$. Suppose that there are at most $p - 1$ anti-edges among the vertices $\{v_1, \cdots, v_p\}$, and let G' be the graph obtained from G by applying the struction operation to the vertex v_0. Then $\tau(G') \leq \tau(G) - 1$.*

Two possible scenarios in which the operation will be applied are illustrated in Figure 1. We will assume that we have a subroutine called **Struction()** that applies the struction operation to a vertex v in G whenever this vertex meets the conditions in Lemma 1.

Let I be an independent set in a graph and let $H = N(I)$. The structure (I, H) is called a *crown* [13], if there exists a matching in G that matches H into

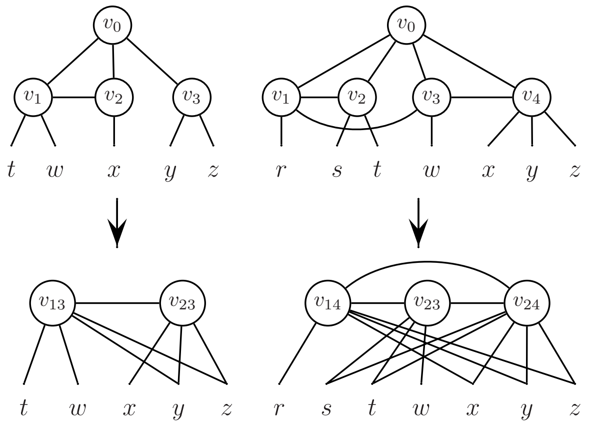


Fig. 1. The struction operation

I. Note that this implies that $|H| \leq |I|$. The graph G is said to be *crown-free* if G does not contain any non-trivial crown [9]. It was shown in [9] that G is crown-free if and only if the NT-decomposition of G is trivial. Moreover, it is also well-known [2] that the NT-decomposition yields a non-trivial crown structure when the decomposition itself is non-trivial.

Next we present an operation that generalizes the folding operation introduced in [7].

Lemma 2. *Let I be an independent set in G and let $N(I)$ be the set of neighbors of I. Suppose that $|N(I)| = |I| + 1$, and that for every subset $\emptyset \neq S \subseteq I$ we have $|N(S)| \geq |S| + 1$.*

1. *If the graph induced by $N(I)$ is not an independent set, then there exists a minimum vertex cover in G that includes $N(I)$ and excludes I.*
2. *If the graph induced by $N(I)$ is an independent set, let G' be the graph obtained from G by removing $I \cup N(I)$ and adding a vertex u_I, then connecting u_I to every vertex $v \in G'$ such that v was a neighbor of a vertex $u \in N(I)$ in G. Then $\tau(G') = \tau(G) - |I|$.*

Let us call a structure $(I, H = N(I))$ satisfying the conditions in Lemma 2 an *almost-crown* structure.

Proposition 2. *Let (G, k) be an instance of* **VC** *such that $|G| \leq 2k$. Then in $O(k^3\sqrt{k})$ time we can reduce (G, k) to an instance (G', k') with $|G'| \leq |G|$ and $k' \leq k$, such that G' is crown-free, or equivalently G' is kernelized ($|G'| \leq 2k'$), and such that an almost-crown structure in G' has been determined in case such a structure exists.*

We will refer to the operation described in Lemma 2 by the *general folding* operation. Two scenarios in which this operation is applicable are given in Figure 2.

We will assume that we have a subroutine called **General-Fold()** that searches for a structure in the graph to which the general folding operation applies, and applies the operation to it in case it exists. We always reduce the graph to a crown-free graph while searching for an almost-crown structure in G. Therefore, if the subroutine **General-Fold()** is not applicable to the graph, i.e., if its application does not change the structure of the graph, then we can assume that the graph is both crown-free and *almost-crown free* (i.e., does not contain an almost-crown).

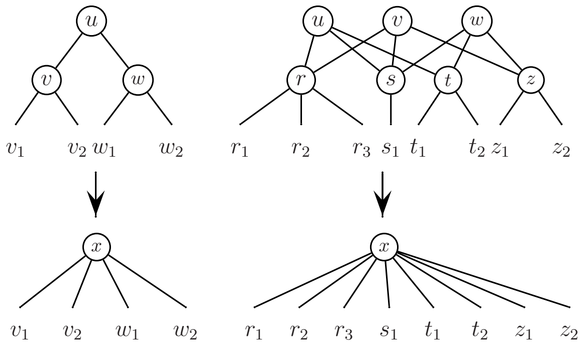


Fig. 2. General folding

3 The Algorithm

The main algorithm is a branch-and-search process. Each stage of the algorithm starts with an instance (G, k) of VC, and tries to reduce the parameter k by identifying a set S of vertices that are entirely contained in a minimum vertex cover of G, and including the vertex set S in the objective minimum vertex cover, which will be called *the partial cover* (or simply the cover) for G, then recursively works on the reduced instances. We will assume that we have the subroutine **General-Fold(G)** described above, and the subroutine **Struction(G)** which applies the struction operation to G.

If a vertex set S is identified such that either there is a minimum vertex cover containing the entire S or there is a minimum vertex cover containing no vertex in S, then we can *branch on the set S*. This means that the algorithm constructs two instances of the VC problem, one by including the set S in the partial cover and the other by excluding the set S from the partial cover, and in the latter case, every vertex that is adjacent to a vertex in S should be included in the partial cover. The algorithm then recursively works on the two reduced instances. If the set S consists of a single vertex v, then we simply say we *branch on v*.

Definitions and Preliminaries

Proposition 3. *Let v be a vertex in G. Then there exists a minimum vertex cover for G containing $N(v)$ or at most $|N(v)| - 2$ vertices from $N(v)$.*

Proposition 4. *Let u and v be two adjacent vertices in G. Then there exists a minimum vertex cover for G that includes v or that excludes v and excludes at least another neighbor of u.*

A vertex u is said to be *dominated* by a vertex v, or alternatively, a vertex v is said to *dominate* a vertex u, if (u, v) is an edge in G and $N(u) \subseteq N[v]$. A vertex u is said to be *almost-dominated* by a vertex v, or alternatively, a vertex v is said to *almost-dominate* a vertex u, if u and v are non-adjacent and $|N(u) - N(v)| \leq 1$.

Proposition 5. *Let u and v be two vertices in G such that v dominates u. Then there exists a minimum vertex cover of G containing v.*

We define next a structure that allows for efficient branching. A *good pair* is a pair of vertices $\{u, z\}$ chosen as follows. For a vertex u in G with neighbors $\{u_1, \cdots, u_d\}$, define its *tag*, denoted $tag(u)$, to be the vector $\eta = \langle \eta_1, \cdots, \eta_d \rangle$, where η_1 is the degree of the largest-degree neighbor of u, η_2 is the degree of the second largest-degree neighbor of u, ..., and η_d is the degree of the smallest-degree neighbor of u. To choose the first vertex in a good pair, we pick a vertex u of minimum degree in G such that the following conditions are satisfied in their respective order.

(i) The vector $tag(u)$ is maximum in lexicographic order over $tag(w)$ for every w in G with the same degree as u.
(ii) If G is regular, then the number of pairs of vertices $\{x, y\} \subseteq N(u)$ such that y is almost-dominated by x is maximized.
(iii) The number of edges in the subgraph induced by $N(u)$ is maximized.

Having chosen the first vertex u in a good pair, to choose the second vertex, we pick a neighbor z of u such that the following conditions are satisfied in their respective order.

(a) If there exist two neighbors of u, say v and w, such that v is almost-dominated by w, then z is almost-dominated by a neighbor of u.
(b) The degree of z is maximum among all neighbors of u satisfying part (a) above. (Note that if no vertex in $N(u)$ is almost-dominated by another vertex in $N(u)$, then (a) is vacuously satisfied by every vertex in $N(u)$, and z will be a neighbor of u of maximum degree.)
(c) The degree of z in the subgraph induced by $N(u)$ is minimum among all vertices satisfying (a) and (b) above. (That is, z is adjacent to the least number of neighbors of u.)
(d) The number of shared neighbors between z and a neighbor of u is maximized over all neighbors of u satisfying (a), (b), and (c) above.

Tuples

Tuples will play a very crucial role in the algorithm by helping to reduce the search space. We define the notion of tuples next and describe how they will be updated and processed by the algorithm.

Definition and intuition. A *tuple* is a pair (S, q) where S is a set of vertices and q is an integer. The tuple will represent the information that in the instance of the problem (G, k) we can look for a minimum vertex cover for G excluding at least q vertices from S. This information will help the algorithm prune the search tree. The algorithm will only consider tuples (S, q) with $q \leq 2$, so we will only focus on such tuples here. A tuple (S, q), where $S = \{u, v\}$, is called a *2-tuple* if it satisfies the following conditions: (1) $q = 1$, (2) $d(u) \geq d(v) \geq 1$, and (3) u and v are non-adjacent. A 2-tuple $(\{u, v\}, 1)$ is a *strong-2-tuple* if it satisfies the additional condition: $d(u) \geq 4$ and $d(v) \geq 4$, or $2 \leq d(u) \leq 3$ and $2 \leq d(v) \leq 3$.

To see how tuples can be used to prune the search space, suppose that the algorithm branches on a vertex z with a set of neighbors $N(z)$. By Proposition 3, there exists a minimum vertex cover in G that contains $N(z)$, or that excludes at least two vertices from $N(z)$. Therefore, when the algorithm branches on z, on the side of the branch where z is included, we can restrict our search to a minimum vertex cover that excludes at least two neighbors of $N(z)$, and we know that this is safe because if such a minimum vertex cover does not exist, then on the other side of the branch where $N(z)$ has been included the algorithm will still be able to find a minimum vertex cover. Consequently, on the side of the branch where z is included, we can work under the assumption that at least two vertices in $N(z)$ must be excluded. This working assumption will be stipulated by creating the tuple $(N(z), q = 2)$. This information will be used by the algorithm to render the branching more efficient. Similarly, if the algorithm branches on a vertex z with a neighbor u, by Proposition 4, either there exists a minimum vertex cover in G that includes z, or there exists a minimum vertex cover in G that excludes z and excludes at least another neighbor of u. Therefore, on the side of the branch where z is excluded, we can restrict our search to a minimum vertex cover that excludes at least two vertices in $N(u)$ (z and another vertex in $N(u)$). This working assumption can be stipulated by creating the tuple $(N(u), q = 2)$. Note that after the removal of $z \in N(u)$ from the graph, the created tuple $(N(u), q = 2)$ will be updated as discussed in the next section.

Updating tuples. Let (S, q) be a tuple. If $q = 0$ then the tuple S will be removed because the information represented by (S, q) is satisfied by any minimum vertex cover. If one of the vertices in S is removed and is excluded from the cover, then the tuple is modified by removing the vertex from S and decrementing q by 1. The correctness of this step can be seen as follows. Suppose that a vertex $u \in S$ has been excluded from the cover. If there exists a minimum vertex cover C that excludes at least q vertices from S, then C excludes at least $q - 1$ vertices from $S - \{u\}$. Therefore the above update to the tuple is valid. If a vertex $u \in S$ is removed from the graph by including it in the cover, the vertex is removed from S and q is kept unchanged. The justification of this step follows from the argument that if there exists a minimum vertex cover C that includes u and excludes at least q vertices from S, then C must exclude q vertices from $S - \{u\}$ (note that the validity of the inclusion of u in the cover is taken care of by the correctness of the steps performed by the algorithm when it includes u in the cover).

Since a tuple imposes certain constraints on the minimum vertex cover sought, one needs to be careful that the constraints imposed by the creation of a tuple do not conflict with the conditions imposed by other operations of the algorithm. The other operations that do impose constraints on the minimum vertex cover sought are the creation of (other) tuples, the struction operation, and the general folding operation. For example, the general folding operation assumes that when we are looking for a minimum vertex cover, we can look for one that either contains the set I or the set $N(I)$ in the structure $(I, N(I))$. This is mainly the reason why the set $N(I)$ can be folded. If the general folding operation is applied, then this constraint imposed by the operation on the minimum vertex cover might conflict with the constraints imposed by a certain tuple. Therefore, to be on the safe side, when we decide to apply the struction or the general folding operations, we will invalidate all the constraints imposed by the tuples. That is, we will basically remove all the tuples. The decision on whether to apply the general folding or the struction operations will be based on the reduction in the parameter resulting from applying these operations. Therefore, we will have two subroutines **Conditional_Struction** and **Conditional_General_Fold** that will apply the struction and general folding operations, respectively. These subroutines will be applied when the gain (reduction in the parameter) resulting from the application of either operation surpasses that resulting from branching on a certain tuple (in case it exists), which will be invalidated after the execution of these operations.

The tuples need to be updated as described above after each operation of the algorithm. We will assume that this step is performed implicitly by the algorithm after each operation.

Storing and branching on 2-tuples. When the algorithm creates tuples it will use them to generate 2-tuples using very simple rules described in steps a.2 and a.3 of the subroutine **Reducing** in Figure 3. Steps a.2 and a.3 of **Reducing** disintegrate a tuple into smaller tuples. During this process, some vertices might be determined to be in a minimum vertex cover by step a.4 of **Reducing**. For example, if $(S = \{u, w, z\}, 1)$ is a tuple, then this tuple imposes the constraint that we can look for a minimum vertex cover that excludes at least one vertex from S. Now if a vertex v is a common neighbor of u, w, and z, then v can be included in a minimum vertex cover satisfying the constraint imposed by the tuple because one of the vertices in S has to be excluded from such a cover. Therefore v will be included by step a.4. Since steps a.2 and a.3 derive more tuples from the tuple S, we need to make sure that the constraints imposed by the tuples generated in these two steps are consistent.

The algorithm, however, creates new tuples when branching. Therefore, if we maintain existing tuples, then the constraints imposed by the newly generated tuples may conflict with those imposed by existing ones. To overcome this hurdle, and since the algorithm only processes 2-tuples, when the subroutine **Reducing** finishes processing the tuples in step a, we will maintain only one 2-tuple and invalidate the rest. Therefore, if 2-tuples exist after step a of **Reducing**, we will pick any strong 2-tuple in case a strong 2-tuple exists and invalidate the rest, or we will pick any 2-tuple and invalidate the rest, otherwise. Since when the algorithm branches it considers 2-tuples first (if they exist), this ensures that when the algorithm creates a new tuple in the next branch, it will have destroyed the only existing tuple when it branched on it. Therefore, after step a of **Reducing**, we will assume that at most one 2-tuple exists.

The algorithm only processes 2-tuples of the form $(S, 1)$. A 2-tuple of the form $(\{u, z\}, 1)$ stipulates that at least one vertex in $\{u, z\}$ must be excluded from the cover. This means that if u is included in the cover then z should be excluded, and hence $N(z)$ must be included; similarly, if z is included in the cover then u should be excluded, and $N(u)$ must be included. Let $(S = \{u, z\}, 1)$ be a 2-tuple. When the algorithm branches on a vertex in this two tuple, this vertex is picked as follows. If there is a vertex $w \in S = \{u, z\}$ such that w has a neighbor u' where u' is almost-dominated by the vertex in $S - \{w\}$, then the algorithm will branch on the vertex in $S - \{w\}$ (that is, if there is a vertex in S with a neighbor that is almost-dominated by the other vertex in S, then the algorithm will pick the other vertex in S). Otherwise, it will pick a vertex in S arbitrarily and branch on it. Without loss of generality, we will always assume that the vertex in the 2-tuple $S = \{u, z\}$ that the algorithm branches on is z. The algorithm can be made anonymous to this choice by ordering the vertices in a 2-tuple as described above whenever the 2-tuple is created.

The Algorithm VC

A tuple, a good pair, or a vertex of degree at least seven, will be referred to by the word *structure*. The algorithm will maintain a list of structures $\mathcal{T}$, and then it will pick a structure and processes it. The structures in $\mathcal{T}$ will be considered in a certain (sorted) order according to their priorities. We will assume that the algorithm implicitly updates the structures in $\mathcal{T}$ and their priorities after each operation. We give below a comprehensive list of the structures Γ that can exist at a certain point in $\mathcal{T}$ listed in a non-increasing order of their priorities.

1 Γ is a strong 2-tuple.
2 Γ is a 2-tuple.
3 Γ is a good pair (u, z) where $d(u) = 3$ and the neighbors of u are degree-5 vertices such that no two of them share any common neighbors besides u.
4 Γ is a good pair (u, z) where $d(u) = 3$ and $d(z) \geq 5$.
5 Γ is a good pair (u, z) where $d(u) = 3$ and $d(z) \geq 4$.
6 Γ is a good pair (u, z) where $d(u) = 4$, u has at least three degree-5 neighbors, and the graph induced by $N(u)$ contains at least one edge (i.e., there is at least one edge among the neighbors of u).
7 Γ is a good pair (u, z) where $d(u) = 4$ and all the neighbors of u are degree-5 vertices such that no two of them share a neighbor other than u.
8 Γ is a vertex z with $d(z) \geq 8$.
9 Γ is a good pair (u, z) where $d(u) = 4$ and $d(z) \geq 5$.
10 Γ is a good pair (u, z) where $d(u) = 5$ and $d(z) \geq 6$.

```
VC($G, \mathcal{T}, k$)
  Input: a graph $G$, a set $\mathcal{T}$ of tuples, and a positive integer $k$.
  Output: the size of a minimum vertex cover of $G$ if the size is bounded by $k$;
          report failure otherwise.

  0. if $|G| > 0$ and $k = 0$ then reject;
  1. apply Reducing;
  2. pick a structure $\Gamma$ of highest priority;
  3. if ($\Gamma$ is a 2-tuple $(\{u,z\},1)$) or ($\Gamma$ is a good pair $(u,z)$ where $z$ is
       almost-dominated by a vertex $v \in N(u)$) or ($\Gamma$ is a vertex $z$ with $d(z) \geq 7$)
     then return
       $\min\{1+$VC$(G-z, \mathcal{T} \cup (N(z),2), k-1),\ d(z)+$ VC$(G-N[z], \mathcal{T}, k-d(z))\}$;
     else   /* $\Gamma$ is a good pair $(u,z)$ where $z$ is not almost-dominated by by any
              vertex in $N(u)$ */
       return
       $\min\{1+$VC$(G-z, \mathcal{T}, k-1),\ d(z)+$ VC$(G-N[z], \mathcal{T} \cup (N(u),2), k-d(z))\}$;

Reducing
  a. for each tuple $(S,q) \in \mathcal{T}$ do
       a.1. if $|S| < q$ then reject;
       a.2. for every vertex $u \in S$ do $\mathcal{T} = \mathcal{T} \cup \{(S-\{u\}, q-1)\}$;
       a.3. if $S$ is not an independent set then
              $\mathcal{T} = \mathcal{T} \cup (\bigcup_{(u,v)\in E, u,v \in S}\{(S-\{u,v\}, q-1)\})$;
       a.4. if there exists $v \in G$ such that $|N(v) \cap S| \geq |S| - q + 1$ then
              return $(1+$VC$(G-v, \mathcal{T}, k-1))$; exit;
  b. if Conditional_General_Fold($G$) or Conditional_Struction($G$) in the
       given order is applicable then apply it; exit;
  c. if there are vertices $u$ and $v$ in $G$ such that $v$ dominates $u$ then
       return $(1+$ VC$(G-v, \mathcal{T}, k-1))$; exit;

Conditional_General_Fold
  if there exists a strong 2-tuple $(\{u,z\},1)$ in $\mathcal{T}$ then
     if the repeated application of General_Fold reduces the parameter by at
       least 2 then apply it repeatedly;
     else if the application of General-Fold reduces the parameter by 1 and
          $(d(u) < 4)$
          then apply it until it is no longer applicable;
  else apply General-Fold until it is no longer applicable;

Conditional_Struction
  if there exists a strong 2-tuple $\{u,v\}$ in $\mathcal{T}$ then
       if there exists $w \in \{u,v\}$ such that $d(w) = 3$ and the Struction is
          applicable to $w$ then apply it;
  else if there exists a vertex $u \in G$ where $d(u) = 3$ or $d(u) = 4$ and such that
          the Struction is applicable to $u$ then apply it;
```

Fig. 3. The algorithm VC

11 Γ is a vertex z such that $d(z) \geq 7$.
12 Γ is any good pair other than the ones appearing in 1–11 above.

The above list gives the structures that could exist in $\mathcal{T}$ and their respective priorities. Moreover, the above list is comprehensive in the sense that for any non-empty graph G, G must contain one of the structures listed above, and the algorithm will have a structure to process.

The algorithm will return the size of a minimum vertex cover in case this size is bounded by k, or otherwise it will reject. The algorithm can be easily modified to return the desired minimum vertex cover itself in case it has size bounded by k. We present the algorithm and prove its correctness next, and we analyze its running time in the next section. The algorithm is given in Figure 3. Note that the algorithm performs *only* two branches *regardless* of the structure picked, which are the ones given in step 3 of the algorithm.

Theorem 1. *The algorithm* **VC** *is correct.*

4 Analysis of the Algorithm

Since the algorithm is a branch-and-bound process, its execution can be depicted by a search tree. The running time of the algorithm is proportional to the number of leaves in the search tree, multiplied by the time spent along each such path. Therefore, the main step in the analysis of the algorithm is deriving an upper bound on the number of leaves in the search tree. We have the following theorem whose proof is inductive and lengthy.

Theorem 2. *The number of leaves in the search tree of the algorithm* **VC** *on an instance* (G, k) *where* G *is a connected graph is upper bounded by* 1.2738^k.

Theorem 3. *The* VC *problem can be solved in* $O(1.2738^k + kn)$ *time.*

References

1. R. Balasubramanian, M. Fellows, and V. Raman. An improved fixed parameter algorithm for Vertex Cover. *Information Processing Letters*, 65:163–168, 1998.
2. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the Weighted Vertex Cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.
3. J. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22:560–572, 1993.
4. L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003.
5. L. Chandran and F. Grandoni. Refined memorisation for vertex cover. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation*, volume 3162 of *Lecture Notes in Computer Science*, pages 61–70, 2004.
6. J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. Taillon. Solving large FPT problems on coarse grained parallel machines. *Journal of Computer and System Sciences*, 67(4):691–706, 2003.

7. J. Chen, I. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.
8. J. Chen, L. Liu, and W. Jia. Improvement on Vertex Cover for low degree graphs. *Networks*, 35:253–259, 2000.
9. M. Chlebik and J. Chlebikova. Crown reductions for the minimum weighted vertex cover problem. In *Electronic Colloquium on Computational Complexity, Report No. 101*, 2004.
10. R. Downey and M. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992.
11. R. Downey and M. Fellows. *Parameterized Complexity*. Springer, New York, 1999.
12. Ch. Ebengger, P. Hammer, and D. de Werra. Pseudo-boolean functions and stability of graphs. *Annals of Discrete Mathematics*, 19:83–98, 1984.
13. M. Fellows. Blow-ups, win/win's and crown rules: some new directions in FPT. In *29th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 2880 of *Lecture Notes in Computer Science*, pages 1–12, 2003.
14. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
15. R. Impagliazzo and R. Paturi. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.
16. G. Nemhauser and L. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.
17. R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 561–570, 1999.
18. R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47:63–77, 2003.
19. J. M. Robson. Algorithms for maximum independent set. *Journal of Algorithms*, 6:425–440, 1977.
20. U. Stege and M. Fellows. An improved fixed-parameter-tractable algorithm for Vertex Cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.

On Comparing Sums of Square Roots of Small Integers[⋆]

Qi Cheng

School of Computer Science
The University of Oklahoma
Norman, OK 73019, USA
qcheng@cs.ou.edu

Abstract. Let k and n be positive integers, $n > k$. Define $r(n,k)$ to be the minimum positive value of

$$|\sqrt{a_1} + \cdots + \sqrt{a_k} - \sqrt{b_1} - \cdots - \sqrt{b_k}|$$

where $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ are positive integers no larger than n. It is an important problem in computational geometry to determine a good upper bound of $-\log r(n,k)$. In this paper we prove an upper bound of $2^{O(n/\log n)}$, which is better than the best known result $O(2^{2k}\log n)$ whenever $n \leq ck\log k$ for some constant c. In particular, our result implies an algorithm *subexponential* in k (i.e. with time complexity $2^{o(k)}(\log n)^{O(1)}$) to compare two sums of square roots of integers of value $o(k\log k)$.

1 Introduction

In computational geometry, one often needs to compare lengths of two polygonal paths, whose nodes are on a integral lattice, and edges are measured according to the L_2 norm. The problem can be reduced to the problem of comparing two sums of square roots of integers. Most work in computational geometry assumes a model of real-number machines, where one memory cell can hold one real number. It is assumed that an algebraic operation, taking a square root as well as a comparison between real numbers can be done in one operation. There is a straight-forward way to compare sums of square roots in real-number machines. But this model is not realistic, as shown in [6,5].

If we consider the problem in the model of Turing machine, then we need to design an algorithm to compare two sums of square roots of integers with low bit complexity. One approach would be approximating the sums by decimal numbers up to a certain precision, and then hopefully we can learn which one is larger. Formally define $r(n,k)$ to be the minimum positive value of

$$|\sqrt{a_1} + \cdots + \sqrt{a_k} - \sqrt{b_1} - \cdots - \sqrt{b_k}|$$

⋆ This research is partially supported by NSF Career Award CCR-0237845.

where $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ are positive integers no larger than n. The time complexity of the approximation approach depends directly on $-\log r(n,k)$, since an approximation of a sum of square roots of integers can be computed in time polynomial in the number of digits in the approximation. One would like to know if $-\log r(n,k)$ is bounded from above by a polynomial function in k and $\log n$. If so, the approximation approach to compare two sums of square root of integers runs in polynomial time. Note that even if the lower bound of $-\log r(n,k)$ is exponential, it does not necessarily rule out a polynomial time algorithm.

Although this problem was put forward during the 1980s [3], progress has been scarce. In [1], it is proved that

$$-\log r(n,k) = O(2^{2k} \log n)$$

using the root separation method. This immediately gives us a polynomial time algorithm of comparing sum of square roots if k is fixed. Qian and Wang [4] gave a constructive upper bound of $r(n,k)$ at $O(n^{-2k+\frac{3}{2}})$, which corresponds to a lower bound of

$$-\log r(n,k) = \Omega(k \log n).$$

They conjecture that $-\log r(n,k) = \Theta(n^{\frac{1}{2}-2^{k-2}})$.

There is a wide gap between the known upper bound and lower bound of $-\log r(n,k)$. Until the fundamental problem has been resolved, we can not even put the presumably easy problem such as Euclidean Minimum Spanning Tree problem in P, and the Euclidean Traveling Salesman problem in NP.

1.1 Our Contribution

From the known upper bound of $-\log r(n,k)$, we conclude that there is a polynomial time algorithm to compare sum of square roots if k is fixed. In this note, we consider the case in the other end of the spectrum when k grows (almost) linear with n.

Definition 1. *An integer n is called square free, if there does not exist a prime p such that p^2 divides n.*

It is well known that there are about $\frac{6n}{\pi^2} + O(\sqrt{n})$ many square free integers less than n. If $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ are distinct square free integers, then their square roots are linearly independent over the field of rational number $\mathbf{Q}$. So it is possible that k and n are linearly related. This case is also practically interesting. We often need to compare paths whose nodes are on an $l \times l$ integral grid. The distance between the lattice points are square roots of integers of size $O(l^2)$. There are l^2 many nodes in the grid, and if we select a dense subset out of the grid points, we arrive in the situation where n is linear in k.

We obtain a lower bound of difference of two sums of square roots. Our lower bound beats the root separation bound as long as $n \leq ck \log k$ for some constant c. The corresponding upper bound of

$$-\log r(n,k) = 2^{O(n/\log n)}$$

becomes *subexponential* in k when $n = o(k \log k)$, or more generally, if the square free parts of the numbers grow at rate $o(k \log k)$. Our bound implies a subexpontial algorithm, i.e., an algorithm with time complexity $2^{o(k)}(\log n)^{O(1)}$, to compare two sums of square roots of small integers. The proof is also simple.

We begin the presentation of our result by defining the notion of multiplicative generators.

Definition 2. *Given two sets of positive integers A and B, we say that B multiplicatively generates A if any number in A can be written as a product of numbers from B with repetition allowed.*

It is easy to see that A multiplicatively generates itself, but for many sets, there exist much smaller sets which multiplicatively generate them. For example, all the square free number less than n are generated by the set of primes less than n, whose cardinality is $O(n/\log n)$.

Theorem 1. *(Main) Let $c_1, c_2, \cdots, c_k, d_1, d_2, \cdots, d_k$ be positive integers. Let*

$$A = \{a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k\}$$

be the set of $2k$ positive square free integers. Assume that $c_i^2 a_i \le n$ for all $1 \le i \le k$ and $d_i^2 b_i \le n$ for all $1 \le i \le k$. Let B be a set which multiplicatively generates A. Then

$$|c_1\sqrt{a_1} + \cdots + c_k\sqrt{a_k} - d_1\sqrt{b_1} - \cdots - d_k\sqrt{b_k}| > (2k\sqrt{n})^{-2^{|B|}+1}.$$

Since A generates itself, so this result recovers the best known lower bound on $r(n, k)$. In many cases, this result improves that bound, since $|B|$ can be smaller than $|A| = 2k$. It is possible that the cardinality of B can be as small as $O(\log k)$, in which case, there is a polynomial time algorithm comparing two sums of square roots.

Our result shows that the multiplicative structure of A affects the minimum possible value of $|c_1\sqrt{a_1} + c_2\sqrt{a_2} + \cdots + c_k\sqrt{a_k} - d_1\sqrt{b_1} - d_2\sqrt{b_2} - \cdots - d_k\sqrt{b_k}|$, which appears to be unknown before. In particular, we show that the root separation lower bound $2^{O(k)} \log n$ of $-\log r(n, k)$ is *not* tight, at least, when n is linear in k. It is still possible that when n is much larger than k, the root separation bound becomes tight. Our result indicates that to achieve the root separation bound, it is important to select the numbers $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ such that they are pairwise relatively prime.

2 The Proof

Let $F = \mathbf{Q}(x_1, x_2, \cdots, x_m)$ be the function field over $\mathbf{Q}$ with indeterminate $x_1, x_2, \cdots, x_m$. Consider a field extension $K = \mathbf{F}[y_1, y_2, \cdots, y_m]/(y_1^2 - x_1, \cdots, y_m^2 - x_m)$ of F. It is a linear space of dimension 2^m over $\mathbf{Q}(x_1, x_2, \cdots, x_m)$, one of whose bases is

$$\{B_S = \prod_{i \in S} y_i | S \subseteq \{1, 2, \cdots, m\}\}.$$

The Galois group G of K over F has order 2^m. For any subset S of $\{1, 2, \cdots, m\}$, define $\sigma_S \in G$ recursively as follows:

1. If $S = \emptyset$, σ_S is the identity element.
2. If $|S| = 1$, then

$$\sigma_{\{i\}}(y_j) = \begin{cases} -y_j & \text{if } i = j \\ y_j & \text{if } i \neq j \end{cases}$$

3. If $|S| > 1$, $\sigma_S = \prod_{i \in S} \sigma_{\{i\}}$.

We have $\sigma_{S'}(B_S) = (-1)^{|S' \cap S|} B_S$ and $G = \{\sigma_S | S \subseteq \{1, \cdots m\}\}$

Lemma 1. *Let $\{\alpha_S | S \subseteq \{1, 2, 3, \cdots, m\}\}$ be a set of 2^m integers. The norm of $\sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S B_S$, denoted by*

$$N_{K/F}(\sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S B_S)$$

is a polynomial in $\mathbf{Z}[x_1, x_2, \cdots, x_m]$.

Proof. By definition,

$$\begin{aligned} N_{K/F}(\sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S B_S) &= \prod_{\sigma \in G} \sigma(\sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S B_S) && (1) \\ &= \prod_{S' \subseteq \{1,2,\cdots,m\}} (\sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S \sigma_{S'}(B_S)) && (2) \\ &= \prod_{S' \subseteq \{1,2,\cdots,m\}} (\sum_{S \subseteq \{1,2,\cdots,m\}} (-1)^{|S \cap S'|} \alpha_S B_S). && (3) \end{aligned}$$

The norm must be an element in $F = \mathbf{Q}(x_1, x_2, \cdots, x_m)$. On the other hand, if we expand the product in the right hand side, it reduces to $\sum_{S \subseteq \{1,2,\cdots,m\}} \beta_S B_S$, where $\beta_S \in \mathbf{Z}[x_1, x_2, \cdots, x_m]$ for any $S \subseteq \{1, 2, \cdots, m\}$. Hence we must have $\beta_S = 0$ for $|S| \geq 1$. Thus

$$N_{K/F}(\sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S B_S) = \beta_\emptyset,$$

which is a polynomial in $\mathbf{Z}[x_1, x_2, \cdots, x_m]$.

Define the polynomial

$$\begin{aligned} &f_{\alpha_\emptyset, \alpha_{\{1\}}, \alpha_{\{2\}}, \cdots, \alpha_{\{1,2,\cdots,m\}}}(x_1, x_2, \cdots, x_m) \\ &= N_{K/F}(\sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S B_S) \in \mathbf{Z}[x_1, x_2, \cdots, x_m]. \end{aligned}$$

Now we are ready to prove the main theorem.

Proof. (of the main theorem) Denote $|B|$ by m. Assume that $B = \{h_1, h_2, \cdots, h_m\}$. There is a natural ring homomorphism

$$\psi : \mathbf{Q}[x_1, x_2, \cdots, x_m, y_1, y_2, \cdots, y_m] \to \mathbf{Q}(\sqrt{h_1}, \sqrt{h_2}, \cdots, \sqrt{h_m})$$

by letting $\psi(y_i) = \sqrt{h_i}$ and $\psi(x_i) = h_i$ for $1 \le i \le m$.

Fix an order among all the subsets of $\{1, 2, \cdots, m\}$. Define $B'_S = \prod_{i \in S} h_i$, and define α_S as

$$\alpha_S = \begin{cases} c_j & \text{if } S \text{ is the first set such that } a_j = B'_S \\ -d_j & \text{if } S \text{ is the first set such that } b_j = B'_S \\ 0 & \text{Otherwise} \end{cases}$$

We have

$$c_1\sqrt{a_1} + \cdots + c_k\sqrt{a_k} - d_1\sqrt{b_1} - \cdots - d_k\sqrt{b_k} = \sum_{S \subseteq \{1,2,\cdots,m\}} \alpha_S B'_S$$

and

$$f_{\alpha_\emptyset,\cdots,\alpha_{\{1,2,\cdots,m\}}}(h_1, h_2, \cdots, h_m) = \prod_{S' \subseteq \{1,2,\cdots,m\}} (\sum_{S' \subseteq \{1,2,\cdots,m\}} (-1)^{|S' \cap S|} \alpha_S B'_S)$$

because of the ring homomorphism. The integer

$$f_{\alpha_\emptyset,\cdots,\alpha_{\{1,2,\cdots,m\}}}(h_1, h_2, \cdots, h_m) \neq 0$$

since $\sqrt{a_1}, \sqrt{a_2}, \cdots, \sqrt{a_k}, \sqrt{b_1}, \sqrt{b_2}, \cdots, \sqrt{b_k}$ are linear independent over $\mathbf{Q}$. So

$$| \prod_{S' \subseteq \{1,2,\cdots,m\}} (\sum_{S \subseteq \{1,2,\cdots,m\}} (-1)^{|S' \cap S|} \alpha_S B'_S)| \ge 1$$

Thus

$$|c_1\sqrt{a_1} + \cdots + c_k\sqrt{a_k} - d_1\sqrt{b_1} - \cdots - d_k\sqrt{b_k}| \tag{4}$$
$$\ge \frac{1}{\prod_{|S'| \neq \emptyset}(\sum_{S \subseteq \{1,2,\cdots,m\}} (-1)^{|S' \cap S|} \alpha_S B'_S)} \tag{5}$$
$$\ge \frac{1}{(2k\sqrt{n})^{2^{|B|}-1}}. \tag{6}$$

The proof relies on the fact that the norm is a nonzero integer, thus has absolute value greater than 1. Every factor in the definition of the norm is not too large (less than $2k\sqrt{n}$ in our case), so the smallest factor should not be too small. The technique has been used in several papers, for example, see [2]. The estimation depends primarily on the number of factors in the definition of the norm.

3 A Corollary from the Main Theorem

Theorem 2. *Let $c_1, c_2, \cdots, c_k, d_1, d_2, \cdots, d_k$ be positive integers. Let $a_1, a_2, \cdots,$ a_k, $b_1, b_2, \cdots, b_k$ be distinct square free positive integers less than m. Assume that $c_i^2 a_i \le n$ for all $1 \le i \le k$ and $d_i^2 b_i \le n$ for all $1 \le i \le k$. Then*

$$|c_1\sqrt{a_1} + \cdots + c_k\sqrt{a_k} - d_1\sqrt{b_1} - \cdots - d_k\sqrt{b_k}| > (2k\sqrt{n})^{-2^{O(m/\log m)}}.$$

Proof. It is well known that the number of primes less than m is $O(m/\log m)$. The set of primes less than m generates all the positive integers less than m. The theorem follows from the main theorem.

Corollary 1. $-\log r(n,k) = 2^{O(n/\log n)}$

4 Conclusion Remarks

In this paper, we prove an upper bound of $2^{O(n/\log n)}$ for $-\log r(n,k)$, by exploring the fact that the algebraic degree of sum of $2k$ square free positive integers can be much less than 2^{2k}. We suspect that $2^{O(k/\log k)}\log n$ type of upper bound holds for much larger n, and leave it as an open problem.

References

1. C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27(1):87–99, 2000.
2. Zhi-Zhong Chen and Ming-Yang Kao. Reducing randomness via irrational numbers. *SIAM J. Comput.*, 29(4):1247–1256, 2000.
3. Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke. The open problems project: Problem 33. http://maven.smith.edu/˜orourke/TOPP/.
4. Jianbo Qian and Cao An Wang. How much precision is needed to compare two sums of square roots of integers? Manuscript, 2005.
5. Arnold Schönhage. On the power of random access machines. In *Proc. 6th Internat. Colloq. Automata Lang. Program.*, volume 71 of *Lecture Notes in Computer Science*, pages 520–529. Springer-Verlag, 1979.
6. A. Shamir. Factoring numbers in $O(\log n)$ arithmetic steps. *Information Processing Letters*, 1:28–31, 1979.

A Combinatorial Approach to Collapsing Words

A. Cherubini[1], P. Gawrychowski[2], A. Kisielewicz[2], and B. Piochi[3]

[1] Politecnico di Milano, Department of Mathematics, Milano, Italy
aleche@mate.polimi.it
[2] University of Wrocław, Institute of Computer Science, Wrocław, Poland
kisiel@math.uni.wroc.pl
[3] Università di Firenze, Department of Mathematics, Firenze, Italy
piochi@math.unifi.it

Abstract. Given a word w over a finite alphabet Σ and a finite deterministic automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$, the inequality $|\delta(Q, w)| \leq |Q| - n$ means that under the natural action of the word w the image of the state set Q is reduced by at least n states. The word w is n-collapsing if this inequality holds for any deterministic finite automaton that satisfies such an inequality for at least one word. In this paper we present a new approach to the topic of collapsing words, and announce a few results we have obtained using this new approach. In particular, we present a direct proof of the fact that the language of n-collapsing words is recursive.

1 Introduction

In this paper by an automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ we mean a finite deterministic automaton with the state set Q, the input alphabet Σ, and the transition function $\delta : Q \times \Sigma \to Q$. The action of Σ on Q given by δ will be denoted simply by concatenation: $qa = \delta(q, a)$. This action extends naturally on the action of the words of Σ^* on Q. Given a word $w \in \Sigma^*$, we will be interested in the difference of the cardinalities $|Q| - |Qw|$, called the *deficiency* of the word w with respect to $\mathcal{A}$ and denoted $\mathrm{df}_{\mathcal{A}}(w)$.

Let $n \geq 1$, a word $w \in \Sigma^*$ is called *n-compressing* for $\mathcal{A}$, if $\mathrm{df}_{\mathcal{A}}(w) \geq n$. An automaton $\mathcal{A}$ is *n-compressible*, if there exists an n-compressing word for $\mathcal{A}$. A word $w \in \Sigma^*$ is *n-collapsing* (over Σ), if it is n-compressing for every n-compressible automaton with the input alphabet Σ.

It has been proved in [11] that n-collapsing words always exist, for any Σ and any $n \geq 1$. In [7] it is shown that, over a fixed alphabet Σ, each n-collapsing word is *n-full*, i.e., it contains any word of length n among its subwords. An n-compressible automaton $\mathcal{A}$ is called *proper* ([1]) if no word of length n is n-compressing for it. Thus to check whether a word $w \in \Sigma$ is n-collapsing it is enough to consider only proper n-compressible automata. For other results and connections with the Černý conjecture see [1,3,4,9].

In [1] certain characterizations of 2-compressing words were given by associating to every word a family of finitely generated subgroups in some finitely generated free groups; it was proved that the property of being 2-collapsing is connected with the subgroup indices in this context. A more geometric version of this idea has been developed in [2]. Unfortunately, these characterizations did not allow either to settle the natural complexity problem concerning collapsing words or to generalize to n-collapsing words (cf. remarks in [4]). In [1] the authors ask a few questions in hope to simplify the characterization.

In this paper we answer all these questions in negative. This is done by using another more combinatorial characterization. Our characterization made also possible to solve important complexity problems, and to tackle more general problems. It forms the base for a series of papers in the area (see e.g. [5,6]).

We view an automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ as a set of transformations labelled by letters of Σ rather than as a standard triple. By the *transformations of* $\mathcal{A}$ we mean those transformations of Q that are induced via δ by letters of Σ. Note that to define an automaton it is enough to assign just to any letter of Σ a transformation of Q.

It is not difficult to see that any proper 2-compressible automaton $\mathcal{A}$ has to have both permutation and non-permutation transformations, and the later correspond to letters a with $\mathrm{df}_{\mathcal{A}}(a) = 1$. Thus, for each such non-permutation transformation a there is a uniquely determined state $z \in Q$ which does not belong to the image Qa and two different states $x, y \in Q$ satisfying $xa = ya$; such a transformation will be referred to as a *transformation of type* $\{x, y\}\backslash z$ (read: x, y identified, z missing). Using this notions we classify proper 2 -compressible automata as follows

Proposition 1. *An automaton $\mathcal{A}$ is proper 2-compressible if and only if $\mathcal{A}$ satisfies one of the following conditions:*

(i) there are x, y such that all non-permutation transformations are of the same type $\{x, y\}\backslash x$, and the group of permutations fixes neither the element x nor the set $\{x, y\}$;

(ii) there is x such that each non-permutation transformation is of type $\{x, z\}\backslash x$ for some z, at least two different types occur, and the group of permutations does not fix x;

(iii) there are x, y such that each non-permutation transformations is of type $\{x, y\}\backslash x$ or $\{x, y\}\backslash y$, both the types occur, and the group of permutations does not fix the set $\{x, y\}$.

This classification corresponds closely to the one in [1], where the automata in cases (i) and (ii) are called MONO, and those satisfying (iii) are called STEREO. We shall call MONO1 and MONO2 the automata in the cases (i) and (ii), respectively.

We note that the use of a different language here is connected with a different view and leads to a different characterization theorem; one that allows a natural generalization. In particular, in general case we speak about transformations a of type $\{S_1, \ldots, S_k\}\backslash M$, where S_i are the sets of states having the same image under a, and M is the the set of non-images under a. Our idea is that this information is generally enough to approach problems on collapsing words.

2 New Characterization of 2-Collapsing Words

We wish to show that for a word $w \in \Sigma^*$ being 2-collapsing over an alphabet Σ is equivalent to the nonexistence of nontrivial solutions to certain systems of conditions on permutations. Consider partitions of Σ into blocks, where blocks are intended to represent types of transformations and closely correspond to the role assignments introduced in [1]. A nontrivial partition of Σ with a distinguished block P, $\emptyset \subseteq P \subseteq \Sigma$, will be called a *DB-partition* and will be denote by (P, Υ), where $\Upsilon = \{B_2, \ldots, B_h\}$ is the induced partition of $\Sigma \setminus P$ $(h \geq 2)$. Let w be a 2-full word over Σ. To each subword of w of the form $\alpha v \beta$, where v is a nonempty word whose all letters belong to P (i.e. $v \in P^+$), while $\alpha \notin P$ and $\beta \in B_j$, we assign a permutation condition of the form

$$1v \in \{1, j\},$$

where the letters of P are treated as permutation variables. Thus, the condition means that the image of 1 under the product v of permutations belongs to the set $\{1, j\}$. The resulting set of permutation conditions (containing all conditions corresponding to subwords of w with the properties described above) will be denoted $\Gamma_w(P, \Upsilon)$ and referred to as the system of permutation conditions determined by a word w and a DB-partition (P, Υ). Note that different orderings of blocks in $\{B_2, \ldots, B_h\}$ lead to systems which are "equivalent" in the sense that both of them have or have not non-trivial solutions; so we don't care the orderings of blocks.

We say that this system *has a solution* if there exists an assignment of permutations on a finite set $\{1, 2, \ldots, N\}$ to letters in P such that all the conditions in $\Gamma_w(P, \Upsilon)$ are satisfied. A trivial solution is one with all permutations fixing 1. Also, in the special case when Υ consists of a unique block B_2 (and in consequence, all j's on the right hand side of the conditions are equal 2), a solution with all permutations fixing the set $\{1, 2\}$ is considered trivial. The remaining solutions are *nontrivial*.

A DB-partition $(P, \{B_1, B_2\})$ of Σ (into exactly 3 blocks, with a distinguished block P) will be called a *3DB-partition*. For such a partition, we define an additional system of permutation conditions as follows. To each subword of w of the form $\alpha v \beta$, with $\alpha \in B_i, \beta \in B_j$, $i, j \in \{1, 2\}$, and $v \in P^+$, we assign a permutation condition of the form

$$iv \in \{1, 2\}$$

(the image of i under v belongs to $\{1, 2\}$). The resulting set of permutation conditions will be denoted by $\Gamma'_w(P, \{B_1, B_2\})$. For such a system, a solution in permutations is *nontrivial* if the image of the set $\{1, 2\}$ does not remain fixed under all the permutations.

Theorem 1. *A word $w \in \Sigma^*$ is 2-collapsing if and only if it is 2-full and the following conditions holds:*

(i) $\Gamma_w(P, \Upsilon)$ has no nontrivial solution for any DB-partition (P, Υ) of Σ;

(ii) $\varGamma'_w(P, \{B_1, B_2\})$ *has no nontrivial solution for any 3DB-partition* $(P, \{B_1, B_2\})$ *of* $\varSigma$.

The reader may observe an explicit similarity with the characterization in [1, Theorem 3.3]. Yet while, indeed, there is a correspondence in the general structure, our approach is almost converse: rather then looking into an algebraic structure behind, we reduce the problem to the simplest conditions on permutations.

Proof. Our general idea is to compute the deficiency of a word w proceeding letter by letter and observing if and how the deficiency increase. To give an idea of how this technique works, we sketch the proof of (i).

To prove the ,,only if" part, recall that if w is 2-collapsing, then it is 2-full. By way of contradiction assume that the system $\varGamma_w(P, \varUpsilon)$ has a nontrivial solution for some DB-partition $(P, \varUpsilon)$, and that this solution consists of permutations on a set $Q = \{1, 2, \ldots, n\}$. Let $\mathcal{A}$ be an automaton over $\varSigma$ with the set Q of states, where the letters in P act as the permutations in the solution and the letters in each of blocks $B_i \in \varUpsilon$ act as (arbitrary) transformations of type $\{1, i\}\backslash 1$. Since the solution is nontrivial, the group of permutations does not fix 1, and if $\varUpsilon = \{B_2\}$ then the group of permutations does not fix the set $\{1, 2\}$. Thus, by Proposition 1, in each case $\mathcal{A}$ is a proper 2-compressible MONO automaton.

Now, if w has no subword of the form $\alpha v \beta$, with $v \in P^+$, $\alpha \in B_i$ and $\beta \in B_j$, then $w = v\alpha_{i_1} \ldots \alpha_{i_m} u$ with $v, u \in P^*$ and $\alpha_{i_j} \in \varUpsilon$, hence $Qw = Q - \{1\}$ and $\mathrm{df}_{\mathcal{A}}(w) = 1$, a contradiction.

Now, let $\alpha v \beta$ be a subword of w with $v \in P^+$, $\alpha \in B_i$ and $\beta \in B_j$, and assume that it is the first subword of this type in w. It follows that $w = s\alpha v \beta t$, where $s, t \in \varSigma^*$, with $\mathrm{df}_{\mathcal{A}}(s\alpha) = 1$, and 1 is missing in the image $Qs\alpha$. Since v is nonempty, the permutation condition $1v \in \{1, j\}$ is in $\varGamma_w(P, \varUpsilon)$. It means that 1 is moved into 1 or j, and consequently 1 or j is missing in the image $Qs\alpha v$. Since β identifies 1 and j, $\mathrm{df}_{\mathcal{A}}(s\alpha v\beta) = 1$, and again 1 is missing in the image $Qs\alpha v\beta$. Proceed letter by letter to get that also in this case $\mathrm{df}_{\mathcal{A}}(w) = 1$: contradiction.

To prove the ,,if" part, assume that w is 2-full but not 2-collapsing, i.e. there exists a proper 2-compressible automaton $\mathcal{A}$ over $\varSigma$, with the set of states $Q = \{1, 2, \ldots, n\}$, for which w is not 2-compressing. If $\mathcal{A}$ is of type MONO, then consider the DB-partition of $\varSigma$, where P represent permutations of $\mathcal{A}$, and $B_2, \ldots, B_h$ represent transformations of types $\{1, 2\}\backslash 1$, ..., $\{1, h\}\backslash 1$, respectively (we assume without loss of generality that $x = 1$ is the distinguished state). The fact that w is not 2-compressing for $\mathcal{A}$ means that, computing deficiency letter by letter, after encountering in w the first letter of any B_i, the deficiency decrease by one, but it does not decrease on further letters. The only segments where the deficiency may decrease are those of the form $\alpha v \beta$, with $\alpha \in B_i, \beta \in B_j$ and $v \in P^+$ when $1v \notin \{1, j\}$. Since the deficiency does not decrease on these segments then the permutations satisfy the corresponding conditions $1v \in \{1, j\}$, as required. The solution they form is nontrivial because of respective conditions (i) or (ii) in Proposition 1.

From our approach, one can easily see that classifying automata as MONO and STEREO has no natural generalization, and is only a very special feature of proper 2-compressible automata. Yet, to demonstrate that the problems in [1] have negative answers we need the following corollary, which can be obtained by more detailed proof as above.

Corollary 1. *For a fixed finite alphabet Σ:*

(i) A word w is 2-compressing for each 2-compressible MONO1 *automaton if and only if the system $\Gamma_w(P, \Upsilon)$ has no nontrivial solution for any DB-partition (P, Υ) with $|\Upsilon| = 1$.*
(ii) A word w is 2-compressing for each 2-compressible MONO2 *automaton if and only if the system $\Gamma_w(P, \Upsilon)$ has no nontrivial solution for any DB-partition (P, Υ) with $|\Upsilon| > 1$.*
(iii) A word w is 2-compressing for each 2-compressible STEREO *automaton if and only if the system $\Gamma_w(P, \Upsilon)$ has no nontrivial solution for any 3DB-partition (P, Υ).*

3 Complexity of Recognizing 2-Collapsing Words

If $\Sigma = \{\alpha, \beta\}$, then we have only two DB-partitions, i.e. only two corresponding system of permutation conditions, each in only one variable. Let

$$E_\alpha(w) = \{k \geq 1 \,:\, \beta\alpha^k\beta \text{ is a subword of } w\};$$
$$E_\beta(w) = \{k \geq 1 \,:\, \alpha\beta^k\alpha \text{ is a subword of } w\}.$$

Using Theorem 1 one almost immediately gets the following result, which closely resembles Proposition 3 of [10]:

Lemma 1. *A word $w \in \{\alpha, \beta\}^*$ is 2-collapsing if and only if it is 2-full and for all integers $n \geq 3$, $0 < r < n$ non of the sets $E_\alpha(w)$ and $E_\beta(w)$ modulo n is included in $\{0, r\}$.*

In order to find an algorithm to check whether a word w is 2-collapsing, we obviously may assume that $n \leq N$, where N is the minimum of the second smallest elements in $E_\alpha(w)$ and in $E_\beta(w)$. Since $N < |w|$, this yields the following corollary (suggested in [4]):

Corollary 2. *For a 2-element alphabet Σ, checking whether a word $w \in \Sigma^*$ is 2-collapsing may be done in polynomial time with respect to $|w|$.*

Moreover, having such a simple insight into the problem, a further question arises naturally: can the problem be solved still in polynomial time when the words are given in the compressed form $\alpha^{k_1}\beta^{k_2}\ldots\beta^{k_n}$ (where $\Sigma = \{\alpha, \beta\}$, k_1 and k_n are allowed to be 0, and the numbers are given in the decimal or binary encodings)? Our new approach makes possible to obtain the following

Theorem 2. *For a 2-element alphabet Σ, checking whether a word $w \in \Sigma^*$ is 2-collapsing may be done in polynomial time with respect to the size of w in the compressed form.*

The proof is a little bit tricky. The idea is that although the number N of cases to be checked may be too large with respect to the size of w in the compressed form, one observes that it is enough to check only the prime divisors of $n < N$ and $n = 4$. The number of such divisors is polynomial with respect to the size of w in the compressed form, but the problem still remains how to overcome the fact that factorization into primes is not known to be in P. Here one applies a certain factorization of the product $k_1 \cdot \ldots \cdot k_n$ into larger divisors, which is enough for our purposes.

At this point we would like also to mention two further results which have been obtained in smaller teams, and which use Theorem 1, as a starting point.

Theorem 3 ([5]). *The problem of recognizing* 2*-collapsing words over a fixed alphabet Σ with more than 2 letters is co-NP-complete.*

It seems that the same methods may be used for n-collapsing words, but the number of independent systems of permutation conditions is much larger.

A word w is n-synchronizing if it is n-collapsing for every n-compressible automaton with $n + 1$ states (i.e. brings all the states to a single state; c.f. [1]). Since there are only finitely many n-state automata over a fixed alphabet Σ, one can observe that the problem of recognizing n-synchronizing words can be solved in polynomial time. Yet, we have:

Theorem 4 ([6]). *The general problem of recognizing* 2*-synchronizing words, where the input consists of an alphabet and a word, is co-NP-complete.*

4 Decidability

For some time it was not even clear that the problem of recognizing of n-collapsing words is decidable. In fact, it is the main result announced in [4], where a large sketch of the proof of this fact is given. The full proof in [8] consists of several lemmas and occupies more than 10 pages. We prove this fact in this section.

Theorem 5. *Let w be a word over a fixed alphabet Σ. If w is not n-collapsing, then there exists an n-compressible automaton $\mathcal{A}$ with number of states $|Q| < 3(n-1)|w|$ such that $\mathrm{df}_{\mathcal{A}}(w) < n$.*

Proof. Suppose that w is not n-collapsing, and let $\mathcal{A}' = \langle Q', \Sigma, \delta' \rangle$ be an n-compressible automaton such that $\mathrm{df}_{\mathcal{A}'}(w) < n$. By n-compressibility it follows that there is also a word w' such that $\mathrm{df}_{\mathcal{A}'}(w') \geq n$, and we may assume that $w' = wu$ extends w. Our aim is to construct an automaton $\mathcal{A} = \langle Q, \Sigma, \delta \rangle$ having the same properties, with $Q \subseteq Q'$, small enough.

Let Σ consist of letters α_i and β_i, chosen so that α_i's represent non-permutation transformations of $\mathcal{A}'$, while β_i represent permutations. We may assume that

$$w = \alpha_{i_0}\Gamma_1\alpha_{i_1}\dots\Gamma_d\alpha_{i_d},$$

where each $\Gamma_j = \beta_{t_1}\dots\beta_{t_i}$ is a product of β_i's, possibly empty (permutations at the beginning and at the end may be ignored). We assume that α_i is of type $\{S_{i,1},\dots,S_{i,m_i}\}/M_i$, that is, M_i is the set of non-images under α_i, and $S_{i,1},\dots,S_{i,m_i}$ are the sets having the same image under α_i. Note that we may assume that the total number of elements in $S_{i,1},\dots,S_{i,m_i}$ is less than $2(n-1)$, and $|M_i| \le n-1$ (otherwise $\mathrm{df}_{\mathcal{A}'}(\alpha_i) \ge n$ on the letter α_i alone). Also $m_i \le n-1$, so the number of images $S_{i,j}\alpha_i$ is less than n. Consequently, by rough estimation, the total number of elements in all the subsets M_i and $S_{i,j}$ and in all the images of $S_{i,j}$ does not exceed $4(n-1)s$, where $s = |\Sigma|$.

Now consider the information we need to determine the exact value of the deficiency $\mathrm{df}_{\mathcal{A}'}(w)$. Consider the partial deficiency sets $X_j = Q' \setminus Q'\alpha_{i_0}\dots\alpha_{i_j}$ for all initial segments of w terminating with some α_i; note that $X_0 = M_{i_0}$, and $X_d = Q' \setminus Q'w$. In general

$$M_{i_j} \subseteq X_j \subseteq M_{i_j} \cup X_{j-1}\Gamma_j\alpha_{i_j}.$$

For $x \in X_{j-1}$, it may happen that $x\Gamma_j\alpha_{i_j} \notin X_j$ if and only if there is a suitable $y \in Q'$ with $y\alpha_{i_j} = x\alpha_{i_j}$ and this is determined by the information on the types of α_i's.

Thus, we need only one more partial information in order to compute all X_j's, namely the following one. For each $j \ge 0$, if $\Gamma_j = \beta_{t_1}\dots\beta_{t_j}$, we need to know the values of β_{t_1} on the states $x \in X_{j-1}$, the values of β_{t_2} on the states $x \in X_{j-1}\beta_{t_1}$, and so on; and finally the values of α_{i_j} on the states $x \in X_{j-1}\Gamma_j$. Since $|X_j| < n$ for all j, this yields totally no more than $(n-1)(|w|-1)$ new states involved (recall that $X_0 = M_{i_0}$). As a result, a partial information on δ' involving less than $(n-1)(4s+|w|-1)$ states determines the required properties of the word w. Indeed, it should be clear that we obtain an automaton $\mathcal{A}$ with no more than $5(n-1)|w|$ satisfying $\mathrm{df}_{\mathcal{A}}(w) < n$, while closing in a natural way cycles in permutations and closing the cycles in non-permutations in such way that they preserve the sets M_i's.

To complete the proof of decidability it is enough to observe that essentially the same argument can be applied to the word wu and to recall the fact that the length of u may be bounded from above by a function of n. The latter follows, for example, from the proof of the basic result [11] concerning the very existence of n-collapsing words (c.f. [4, Section 1]). It follows that there exists an n-compressible automaton $\mathcal{A} = \langle Q, \Sigma, \delta\rangle$, with $|Q|$ bounded by a function of n and $|w|$, witnessing that w is not n-collapsing.

To obtain our bound for $|Q|$, first note that we may not need the whole information on the sets $S_{i,1},\dots,S_{i,m_i}$. It is enough to know, for each α_{i_j}, single pairs x, y from some of these sets satisfying suitable equalities $y\alpha_{i_j} = x\alpha_{i_j}$; and we do not even need to know the exact value of the image. Thus, considering α_{i_j}, we need to know M_{i_j} together with one of the following: either the image

$x\alpha_{i_j}$ for $x \in X_{j-1}\Gamma_j$ or an element y with $y\alpha_{i_j} = x\alpha_{i_j}$. Then, one can see that a partial definition of δ involving a subset $S \subseteq Q'$ with $|S| \leq (n-1)(s+|w|-1)$ is enough to guarantee that $\mathrm{df}_{\mathcal{A}}(w) < n$ for any $\mathcal{A} = \langle Q, \Sigma, \delta\rangle$ having such a subset of states and with δ extending the partial definition.

Now, from the fact that $\mathrm{df}_{\mathcal{A}'}(wu) \geq n$, we infer that there are $x, y \in Q'w$ such that $xu = yu$. For $u = \gamma_1 \ldots \gamma_t$, denote $x_0 = x$, and $x_i = x_{i-1}\gamma_i$, and similarly, $y_0 = y$, and $y_i = y_{i-1}\gamma_i$, for all $1 \leq i \leq t$. Note that $x_t = xu = yu = y_t$.

If all $x_i, y_i \in S$, then we may choose $Q = S$, and to finish the proof as before (this part requires some more detailed analysis that it is possible to complete definitions of non-transformations without adding new states while keeping all the necessary properties). The resulting automaton $\mathcal{A} = \langle Q, \Sigma, \delta\rangle$ satisfies $\mathrm{df}_{\mathcal{A}}(w) = k < n$, and $\mathrm{df}_{\mathcal{A}}(w) = k+1$. If $k < n-1$, we simply add new states $q'_1, \ldots, q'_{n-1-k}$, all of them transformed into q'_1 by all the transformations: we obtain so an automaton with the required properties and with less than $|S| + (n-1) = (n-1)(s+|w|)$ states.

If all x_i's are in S, while some of y_i's are not, assume u be the shortest possible with the desired property, to argue that there are at most $(n-1)|w|$ states y_i not in S. Indeed let a be the smallest index for which $y_a\gamma_a \notin S$, and b the largest one with $y_b\gamma_b^{-1} \notin S$. Consider all the pairs $x_i\gamma_i = x_{i+1}$ with $a \leq i < b$. We may assume that there is no i, j such that $x_i = x_j$ and $\gamma_i = \gamma_j$; otherwise we could build u', shorter than u, with the same properties. But in the first part there are no more than $(n-1)|w|$ such pairs with determined images, whence the claim. Adjoin all x_i's and y_i's to S to obtain, as before, the automaon we are looking for, whose set Q of states fulfills $|Q| < (n-1)(s+2|w|)$.

At last, if both some x_i and some y_i are not in S, we may argue that, again, the total number of such states not in S does not exceed $(n-1)|w|$; otherwise we can modify u reducing ourselves to the former case (this part is left to the reader).

Theorem 5 obviously shows that, for each $n > 1$, the language of n-collapsing words over Σ is recursive (it is always enough to check a finite number of automata). The bound in our theorem is even slightly better than $3(n-1)|w|+n+1$ in [8, Theorem 1] (and [4, Theorem 1]). Remark that considering separately some extremal cases (like all the transformations are the same) there is a room to still improve the bound in the theorem.

5 Collapsing Words on MONO and STEREO Automata

To complete the paper we provide the ideas of solutions to the open problems from [1] which we mentioned above.

For a 3-element alphabet Σ it is not difficult to find a word, which is 2-compressing for each 2-compressible STEREO automaton, but fails to be 2-collapsing; the same question for MONO was stated as an open problem (Question 5.1 in [1]), intending to consider possible simplifications of the characterization given there. We give a negative answer to this and other two questions, producing suitable counterexamples.

Proposition 2. *Let $\Sigma = \{\alpha, \beta, \gamma\}$ There is a word $w \in \Sigma^+$ which is 2-compressing for any* MONO *automaton, with input alphabet Σ, and which fails to be 2-collapsing.*

In the proof below we use notation $\alpha = (i_1 i_2 ...)...$ to denote a permutation α which has a cycle sending i_1 into i_2 in its decomposition into a product of disjoint cycles.

Proof. Let A be a finite automaton with the set of states $Q = \{1, 2, 3\}$ and input alphabet Σ. Let α be of type $\{1, 2\}\backslash 1$, β of type $\{1, 2\}\backslash 2$, and $\gamma = (123)$ a permutation. By Proposition 1 A is a proper 2-compressible STEREO automata and it is easy to verify that all the words in $X \cup \{\alpha, \beta\}^+$, with $X = \{\alpha\gamma x, \alpha\gamma^3 x, \beta\gamma^2 x, \beta\gamma^3 x |\ x \in \{\alpha, \beta\}\}$, are not 2-compressing for A.

Now consider the 2-full word

$$w = \beta\alpha\gamma\beta^2\gamma^2\beta\alpha\gamma\alpha\beta\alpha\gamma\beta\alpha\beta^2\gamma^2\alpha\beta\gamma^3\beta\alpha^2\gamma\beta\gamma^2\alpha.$$

All of its subwords of the form $x\gamma^+ y$ with $x, y \in \{\alpha, \beta\}$ are in X (whence it does not compress A and is not 2-collapsing), but it is 2-compressing for each MONO automaton with input alphabet $\{\alpha, \beta, \gamma\}$ by Corollary 1. In fact we shall prove that for any DB-partition (P, Υ) of $\{\alpha, \beta, \gamma\}$ the system $\Gamma_w(P, \Upsilon)$ has no nontrivial solution.

Assume $|P| = 1$ and denote by π the element of P. If $\Upsilon = B_2$, then the solutions of the system $\Gamma_w(P, \Upsilon)$ are all trivial. In fact for all choice of π, the conditions $1\pi \in \{1, 2\}$, $1\pi^2 \in \{1, 2\}$ occur in $\Gamma_w(P, \Upsilon)$: the solutions of the former condition either fix 1 or have the form $(12...)...$, and $\pi = (12...)...$ is a solution of the latter only if the cycle $(12...)$ reduces to (12). If $\Upsilon = \{B_2, B_3\}$, then the solutions of the system $\Gamma_w(P, \Upsilon)$ are still trivial, since for each choice of π the conditions $1\pi \in \{1, 2\}$, $1\pi \in \{1, 3\}$ occur in the system $\Gamma_w(P, \Upsilon)$, hence all the solutions must fix 1.

So assume that $|P| = 2$. Let $\Upsilon = \{\alpha\}$. The system $\Gamma_w(P, \Upsilon)$ is

(1) $1\gamma\beta^2\gamma^2\beta \in \{1, 2\}$, (2) $1\gamma \in \{1, 2\}$, (3) $1\beta \in \{1, 2\}$,
(4) $1\gamma\beta \in \{1, 2\}$, (5) $1\beta^2\gamma^2 \in \{1, 2\}$, (6) $1\beta\gamma^3\beta \in \{1, 2\}$,
(7) $1\gamma\beta\gamma^2 \in \{1, 2\}$,

The solutions of the condition (3) either fix 1 or are of the form $\beta = (12...)...$; similarly the solutions of the condition (ii) either fix 1 or are of the form $\gamma = (12...)...$. Suppose that it is not the case that both of them fix 1. Assume $1\gamma = 1$ and let $\beta = (12x...)...$: hence by condition (5) $x\gamma^2 = 1\beta^2\gamma^2 \in \{1, 2\}$. This yields $\beta = (12)...$; in fact if $x\gamma^2 = 1$ then $x = 1$, else by the first condition we get $x\gamma^2\beta = 2\beta \in \{1, 2\}$. Whence by conditions (6) and (7) $2\gamma^2 = 2 = 2\gamma^3$, thus γ fixes 2 too and the solutions are trivial. So let $\gamma = (12...)...$ Condition (4) yields $2\beta \in \{1, 2\}$, whence either β fixes 1 and 2 or it is of the form $\beta = (12)...$ In both cases by condition (5) it follows that $\gamma = (12)...$ and the solutions are always trivial.

The cases when $\Upsilon = \{\beta\}$ or $\Upsilon = \{\gamma\}$ are similar and still there exist only trivial solutions.

Proposition 2 answers (in negative) to Question 5.1 in [1]. The following Proposition gives a negative answer to Question 5.2.(i) in [1]:

Proposition 3. *Let $\Sigma = \{\alpha, \beta, \gamma\}$. There is a word $w \in \Sigma^+$ which is 2-compressing for any* MONO *automaton with one non-permutation transformation, but not for each* MONO *automaton with two non-permutation transformations.*

Proof. Let A be a finite automaton with set of states $Q = \{1, 2, 3\}$ and input alphabet Σ. Let α be of type $\{1, 2\}\backslash 1$, β of type $\{1, 3\}\backslash 1$, and $\gamma = (123)$ a permutation. A is a proper 2-compressible MONO automaton with 2 non permutation transformations. Consider the set $X = \{x\gamma^2\beta, x\gamma^3\beta, x\gamma\alpha, x\gamma^3\alpha \mid x \in \{\alpha, \beta\}\}$, it is easy to verify that all words in $X \cup \{\alpha, \beta\}^*$ have deficiency 1 with respect to A.

Consider the 2-full word

$$w = \gamma\alpha\beta^2\alpha\gamma^2\beta\gamma\alpha^2\beta\gamma^3\beta\alpha\gamma\alpha\beta\alpha^2\gamma^2\beta\alpha\beta\gamma^2\beta^2\alpha\gamma.$$

All of its subwords of the form $x\gamma^+y$ with $x, y \in \{\alpha, \beta\}$ are in X (whence it does not compress A and is not 2-collapsing), but is 2-compressible with respect to each MONO automaton with input alphabet $\{\alpha, \beta, \gamma\}$ by Corollary 1. In fact for any DB-partition (P, Υ) of Σ the system $\Gamma_w(P, \Upsilon)$ has no nontrivial solution; the proof is analogous to that of Proposition 2.

It is easy to deduce the following plus .1em

Corollary 3. *Let $\Sigma = \{\alpha, \beta, \gamma\}$. There is a word $w \in \Sigma^+$ which is* 2*-compressing for each* MONO1 *automaton, and which fails to be 2-compressing for any* MONO2 *automaton with input alphabet Σ.*

Also Question 5.2.(ii) in [1], regarding STEREO automata, has a negative answer. The proof of the following proposition is similar to the former ones:

Proposition 4. *Let $\Sigma = \{\alpha, \beta, \gamma\,\delta\}$. There is a word $w \in \Sigma^+$ which is 2-compressing for each* STEREO *automaton with input alphabet Σ and with two non-permutation transformations, but fails to be 2-compressing for some* STEREO *automaton with input alphabet Σ with three non-permutation transformations.*

References

1. D. S. Ananichev, A. Cherubini, and M. V. Volkov, *Image reducing words and subgroups of free groups*, Theor. Comput. Sci. 307, no.1 (2003), 7792.
2. D. S. Ananichev, A. Cherubini, and M. V. Volkov, *An inverse automata algorithm for recognizing 2-collapsing words*, in: M. Ito, M. Toyama (eds.), Developments in Language Theory, Lect. Notes Comp. Sci. 2450, Springer, Berlin 2003, 270282.
3. D. S. Ananichev and I. V. Petrov, *Quest for short synchronizing words and short collapsing words*, WORDS. Proc. 4th Int. Conf., Univ. of Turku, Turku, 2003, 411418.
4. D. S. Ananichev, I. V. Petrov, and M. V. Volkov, *Collapsing words: A Progress Report*, in: C. De Felice and A. Restivo (eds.), Developments in Language Theory, Lect. Notes Comp. Sci. 3572, Springer, Berlin 2005, 11-21.
5. A. Cherubini and A. Kisielewicz, *Recognizing collapsing words is co-NP-complete*, Proceedings of DCFS 2006, to appear.

6. P. Gawrychowski and A. Kisielewicz, *Recognizing 2-synchronizing words*, preprint 2006.
7. S. W. Margolis, J.-E. Pin, and M. V. Volkov, *Words guaranteeing minimum image*, Internat. J. Foundations Comp. Sci. 15 (2004) 259276.
8. I. V. Petrov, *An algorithm for recognizing n-collapsing words*, preprint 2005.
9. J.-E. Pin, *On two combinatorial problems arising from automata theory*, Ann. Discrete Math. 17 (1983) 535548.
10. E. V. Pribavkina, *On some properties of the language of 2-collapsing words*, A, in: C. De Felice and A. Restivo (eds.), Developments in Language Theory, Lect. Notes Comp. Sci. 3572, Springer, Berlin 2005, 374-384.
11. N. Sauer and M. G. Stone, *Composing functions to reduce image size*, Ars Combinatoria 1 (1991) 171176.

Optimal Linear Arrangement of Interval Graphs

Johanne Cohen[1], Fedor Fomin[2], Pinar Heggernes[2],
Dieter Kratsch[3], and Gregory Kucherov[4]

[1] LORIA, 54506 Vandoeuvre-lès-Nancy Cedex, France
Johanne.Cohen@loria.fr
[2] Department of Informatics, University of Bergen, 5020 Bergen, Norway
{Fedor.Fomin, Pinar.Heggernes}@ii.uib.no
[3] LITA, Université de Metz, 57045 Metz Cedex 01, France
kratsch@sciences.univ-metz.fr
[4] LIFL/CNRS, 59655 Villeneuve d'Ascq, France
Gregory.Kucherov@lifl.fr

Abstract. We study the optimal linear arrangement (OLA) problem on interval graphs. Several linear layout problems that are NP-hard on general graphs are solvable in polynomial time on interval graphs. We prove that, quite surprisingly, optimal linear arrangement of interval graphs is NP-hard. The same result holds for permutation graphs. We present a lower bound and a simple and fast 2-approximation algorithm based on any interval model of the input graph.

1 Introduction

A *linear layout* (or simply *layout*) of a given graph $G = (V, E)$ is a linear ordering of its vertices. Assuming that the vertices of G are numbered from 1 to n, a layout is a permutation $L(1), L(2), \ldots, L(n)$. The *weight* of a layout L on G is $\mathcal{W}(G, L) = \sum_{(u,v)\in E} |L(u) - L(v)|$. An *optimal linear arrangement* (OLA) of G is a layout with the minimum weight, i.e., $\text{argmin}_L \mathcal{W}(G, L)$. We denote $\mathcal{W}(G) = \min_L \mathcal{W}(G, L)$ and call it the *minimum weight* on G.

Computing the optimal linear arrangement (the OLA problem) is NP-hard [11], and it remains NP-hard for bipartite graphs [6]. The problem is solvable in polynomial time for trees [7,3,19], and for some other restricted graph classes such as grids or hypercubes [4]. There is an approximation algorithm for general graphs with performance ratio $O(\log n)$ [18].

A well-known vertex ordering problem related to OLA is the Bandwidth Minimization problem. The bandwidth of a layout L on G is $b(G, L) = \max_{(u,v)\in E} |L(u) - L(v)|$. The *bandwidth* of G is the minimum bandwidth of any layout of G, i.e., $bw(G) = \min_L b(G, L)$. The bandwidth minimization problem is also NP-hard on general graphs [10]. It remains NP-hard even on the restricted class of trees [17]. Furthermore, for general graphs, bandwidth cannot be approximated by a polynomial time algorithm within a constant factor [21], but it can be approximated in polynomial time with a factor of $O(\log^{9/2} n)$ [9].

It is well known that many NP hard-problems are solvable in polynomial time on interval graphs. In 1985, Johnson wrote in his NP-completeness column: "Indeed, I know of no NP-completeness results for interval graphs, although there are still some possibilities in Table 1, in addition to such naturals as BANDWIDTH and SUBGRAPH ISOMORPHISM" [13]. Interestingly, a bit later, it appeared that the bandwidth minimization problem is solvable in polynomial time for interval graphs. For an interval graph with n vertices given by an interval model, Kleitman and Vohra's algorithm solves the decision problem "Is $bw(G) \leq k$?" in $O(nk)$ time, and it can be used to produce a layout with the minimum bandwidth in $O(n^2 \log n)$ time [14]. Furthermore, Sprague has shown how to implement Kleitman and Vohra's algorithm to answer the decision problem in $O(n \log n)$ time, and thus produce a minimum bandwidth layout in $O(n \log^2 n)$ time [20]. We refer the reader to [4] for a survey of known results on the OLA, bandwidth and other related layout problems.

To our knowledge, optimal linear arrangement of interval graphs has not been studied so far. In this paper, we show that, in contrast to bandwidth minimization, the OLA problem is NP-hard on interval graphs. We also show that the problem can be approximated within a constant factor of 2 by a simple algorithm.

Besides its theoretical interest, the class of interval graphs is widely acknowledged as an important graph class, due to a number of applications. Interval graphs are extensively used in bioinformatics, typically to model the genome physical mapping problem, which is the problem of reconstructing the relative positions of DNA fragments, called *clones*, out of information of their pairwise overlaps (see e.g. [22]). However, interval graphs appear also in other situations in bioinformatics, such as for gene structure prediction for example [1]. In [8], interval graphs are used to model temporal relations in protein-protein interactions. In that paper, an optimal linear arrangement of an interval graph models an "optimal" molecular pathway, and the problem of efficiently computing this arrangement is explicitly raised. This provides a direct motivation for the present study.

This paper is organized as follows. In Section 2, graph notations are introduced. We obtain a lower bound for the minimum weight of a linear arrangement for general graphs in terms of the degrees of the vertices. In Section 3, we prove that the OLA problem is NP-complete for interval graphs. In Section 4, using the lower bound we show that both the left endpoint ordering and the right endpoint ordering of an interval graph are 2-approximations for the Optimal Linear Arrangement problem. In Section 5, we first show that the NP-completeness result holds also for permutation graphs, and then discuss approximation algorithms for OLA of the more general class of cocomparability graphs.

2 Preliminaries

We consider only finite, undirected and simple graphs. For $G = (V, E)$, we will denote $|V|$ as n and $|E|$ as m. We sometimes refer to the vertex set of G as $V(G)$ and the edge set as $E(G)$. We let $N(v)$ denote the set of vertices adjacent to v. The *degree* of a vertex v in graph G, $d_G(v)$, is the number of vertices adjacent to v in G. $\Delta(G)$ denotes the maximum degree of a vertex in graph G. The subgraph of $G = (V, E)$ induced by $V' \subseteq V$ will be referred to as $G[V']$. The complement of a graph G is denoted by $\overline{G}$ and has the same vertex set as G, and $(x, y) \in E(\overline{G})$ if and only if $(x, y) \notin E(G)$.

A layout L of a graph $G = (V, E)$ can be seen as an ordering $(v_1, v_2, \ldots, v_n)$ of V, meaning that $L(v_j) = j$, for $1 \leq j \leq n$. We extend this notation to subsets of vertices. Let $V_1, \ldots, V_i$ be a partition of V. If a layout L of G has the form $(V_1, \ldots V_i)$, then it implies that

- $\forall j, \forall \ell, 1 \leq j < \ell \leq i, \forall u \in V_j, \forall w \in V_\ell, L(u) < L(w)$
- $\forall \ell, 1 \leq \ell \leq i$, the order of L inside V_ℓ is an arbitrary order of V_ℓ.

A graph $G = (V, E)$ is an *interval graph* if there is a one-to-one correspondence between V and a set of intervals of the real line such that, for all $u, v \in V$, $(u, v) \in E$ if and only if the intervals corresponding to u and v have a nonempty intersection. Such a set of intervals $\mathcal{I}$ is called an *interval model* for G. We assume that an interval model is given by a left endpoint and a right endpoint for each interval, namely, $l(v)$ and $r(v)$ for all $v \in V$. Furthermore, we assume that we are also given a sorted list of the endpoints, and that the endpoints are distinct.

First, we study OLA of simple topologies, like stars and complete graphs. A *star*, denoted by S_α, is a tree such that one vertex, called the center, is adjacent to α leaves. A *complete graph*, denoted by K_n, is a graph on n vertices such that all vertices are pairwise adjacent. The following lemmas give the weight of the optimal linear arrangement for these particular topologies.

Lemma 1. *Let K_n be the complete graph on n vertices. Then $\mathcal{W}(K_n) = \frac{(n-1)n(n+1)}{6}$.*

Proof. Straightforward, as all layouts yield the same weight. $\square$

Lemma 2. *Let S_α be the star with a center vertex c and α leaves. Then every layout L of S_α satisfies the following:*

- $\frac{\alpha}{2}(\frac{\alpha}{2} + 1) \leq \mathcal{W}(S_\alpha, L) \leq \frac{\alpha}{2}(\alpha + 1)$ *and* $\mathcal{W}(S_\alpha) = \frac{\alpha}{2}(\frac{\alpha}{2} + 1)$, *if α is even,*
- $(\lfloor \frac{\alpha}{2} \rfloor + 1)^2 \leq \mathcal{W}(S_\alpha, L) \leq (\lfloor \frac{\alpha}{2} \rfloor + 1)\alpha$ *and* $\mathcal{W}(S_\alpha) = (\lfloor \frac{\alpha}{2} \rfloor + 1)^2$, *if α is odd,*

and a permutation L is an optimal linear arrangement if and only if L places c at the middle position.

Proof. Assume that $L(c) = k$. Then $\mathcal{W}(S_\alpha, L) = \sum_{i=1}^{k-1} i + \sum_{i=1}^{\alpha+1-k} i = (k-1)^2 + \frac{\alpha}{2}(\alpha + 3 - 2k)$. For the case where α is even, $\mathcal{W}(S_\alpha, L)$ reaches its minimum for $k = \frac{\alpha}{2} + 1$ or for $k = \frac{\alpha}{2}$. In this case, $\mathcal{W}(S_\alpha) = \frac{\alpha}{2}(\frac{\alpha}{2} + 1)$. Moreover $\mathcal{W}(S_\alpha, L)$ reaches its maximum for $k = 1$ or for $k = \alpha + 1$. The same arguments can be applied for the case where α is odd. $\square$

These results will be needed to prove the NP-completeness of the OLA problem on interval graphs and to give a 2-approximation algorithm for it. The following lower bound for optimal linear arrangement of any graph is obvious, and it will be useful when analyzing the performance ratio of some algorithms.

Lemma 3. *Let $G = (V, E)$ be a graph, $E = E_1 \cup E_2$ and $E_1 \cap E_2 = \emptyset$. Then $\mathcal{W}(G) \geq \mathcal{W}(G_1) + \mathcal{W}(G_2)$, where $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$.*

Corollary 1. *Let $G = (V, E)$, $V = V_1 \cup \cdots \cup V_n$, and $E = E_1 \cup \cdots \cup E_n$, where $E_1, \cdots, E_n$ are pairwise disjoint. Then $\mathcal{W}(G) \geq \mathcal{W}(G_1) + \ldots + \mathcal{W}(G_n)$, where $G_i = (V_i, E_i)$, $1 \leq i \leq n$.*

All these results will be useful to compute the lower and upper bounds of the weight $\mathcal{W}(G, L)$ of a layout L of G. For example, consider a graph G composed of two disjoint complete graphs K_α and K_b and an additional vertex c adjacent to all other vertices of the graph. The set of edges of this graph can be easily partitioned into three sets. From Corollary 1, by construction we have $\mathcal{W}(G) \geq \mathcal{W}(K_b) + \mathcal{W}(K_\alpha) + \mathcal{W}(S_{\alpha+b})$. Moreover, the following layout L of G is considered: $V(K_\alpha), c, V(K_b)$. Layout L has weight $\mathcal{W}(K_b) + \mathcal{W}(K_\alpha) + \mathcal{W}(S_{\alpha+b})$. The previous inequality implies that L is an optimal linear arrangement.

3 The Complexity of the OLA Problem on Interval Graphs

The goal of this section is to prove the following theorem.

Theorem 1. *The problem of deciding, for an interval graph $G = (E, V)$ and a constant K, whether $\mathcal{W}(G) \leq K$ is NP-complete.*

The proof will be by reduction from the 3-PARTITION problem [11]:

3-PARTITION
Instance: A finite set A of $3m$ integers $\{a_1, \ldots, a_{3m}\}$, a bound $B \in Z^+$ such that $\sum_{i=1}^{3m} a_i = mB$.
Question: Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that, for all $1 \leq i \leq m$, $\sum_{a \in A_i} a = B$?

3-PARTITION is known to be NP-complete in the strong sense [11] (Even if B is polynomially bounded by the size of instance, the problem is still NP-complete). Note that we do not require here that each A_i is composed of exactly three elements.

The structure of our proof will be as follows. We first construct a graph $\mathcal{H}(B, m)$ depending on two natural numbers B and m, and we describe the structure of its optimal linear arrangement. In the second part, we describe a polynomial-time reduction from 3-PARTITION, i.e., we encode numbers $\{a_1, \ldots, a_{3m}\}$ by adding some additional edges to graph $\mathcal{H}(B, m)$, and show that an optimal linear arrangement of this extended graph corresponds precisely to a 3-partition of $\{a_1, \ldots, a_{3m}\}$.

For simplicity of notation in our proofs, in this section we will let $\mathcal{K}(n) = \mathcal{W}(K_n)$ and $\mathcal{S}(\alpha) = \mathcal{W}(S_\alpha)$, where K_n is the complete graph on n vertices, and S_α is the star with α leaves.

3.1 Construction of $\mathcal{H}(B, m)$ and Its Optimal Linear Arrangement

Let m and B be two integers. We assume that m is even. The set of vertices of $\mathcal{H}(B, m)$ will be the union of several disjoint sets

$$V(\mathcal{H}(B, m)) = R_1 \cup X \cup V \cup Y \cup Z \cup R_2.$$

The number of vertices in each set is defined as follows.

- Each of R_1 and R_2 has $3m^3(B+1)$ vertices,
- X is the union of disjoint sets $X_1, \ldots, X_{m/2}$, where each X_i has $2(B+1)$ vertices; similarly, Z is the union of disjoint sets $Z_1, \ldots, Z_{m/2}$, where each Z_i has $2(B+1)$ vertices,
- V has $(m+1)$ vertices,
- Y has mB vertices.

The set of edges of $\mathcal{H}(B,m)$ is defined as follows.

- Vertices of $R_1 \cup X$ form a clique, i.e., they are all pairwise adjacent; vertices in R_1 have no other neighbors,
- vertices of $R_2 \cup Z$ form a clique; vertices in R_2 have no other neighbors,
- vertices $V = \{v_1, \ldots, v_{m+1}\}$ form a clique,
- for each $1 \le i \le m/2$, v_i is adjacent to all vertices of $X_i \cup \ldots \cup X_{m/2}$,
- for each $1 \le i \le m/2$, v_{m+2-i} is adjacent to all vertices of $Z_i \cup \ldots \cup Z_{m/2}$,
- each vertex of Y is adjacent to all vertices of V,
- $\mathcal{H}(B,m)$ has no edges other than those defined above.

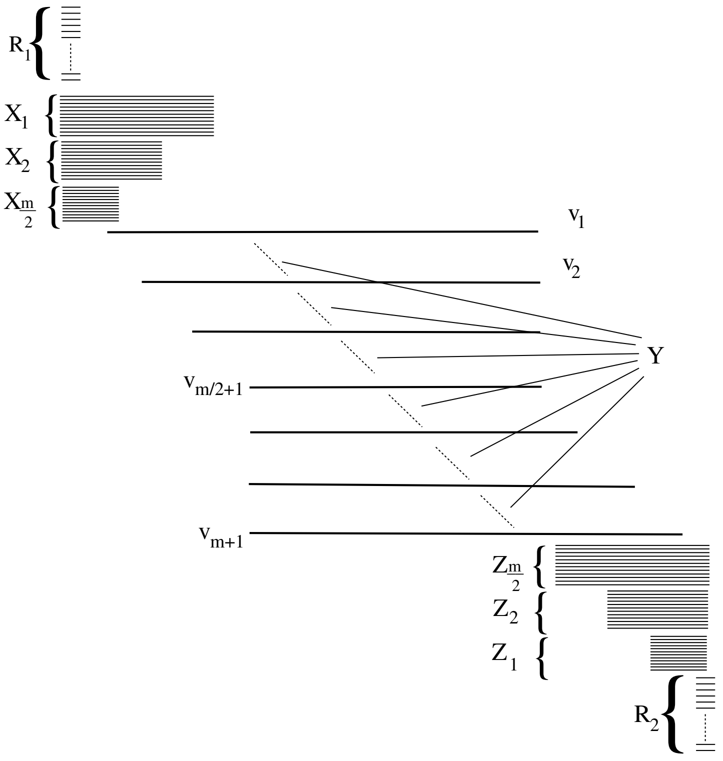


Fig. 1. Interval representation of graph $\mathcal{H}(B,m)$

An interval representation of graph $\mathcal{H}(B,m)$ is given in Figure 1. From this figure, it is clear that $\mathcal{H}(B,m)$ is an interval graph. From Lemma 1, a lower bound on $\mathcal{W}(\mathcal{H}(B,m))$ can be now established as follows.

Lemma 4. $\mathcal{W}(\mathcal{H}(B,m)) \geq 2\mathcal{K}(3m^3(B+1)+m(B+1)) + 2\sum_{i=1}^{m/2} \mathcal{S}(2(m-i+1)(B+1)-m) + \mathcal{S}(mB) + \mathcal{K}(m+1)$.

Proof. Using Corollary 1, we can estimate the lower bound as follows: $\mathcal{W}(\mathcal{H}(B,m)) \geq \mathcal{K}(|R_1|+|X|)+\sum_{i=1}^{m/2} \mathcal{S}(|X_i|+\ldots+|X_{m/2}|+|Y|)+\sum_{i=1}^{m/2} \mathcal{S}(|Z_i|+\ldots+|Z_{m/2}|+|Y|)+\mathcal{K}(|V|)+\mathcal{S}(|Y|)+\mathcal{K}(|Z|+|R_2|)$. Here terms $\mathcal{K}(|R_1|+|X|)$ and $\mathcal{K}(|Z|+|R_2|)$ correspond to complete graphs formed respectively by vertex sets $R_1 \cup X$ and $Z \cup R_2$. Each term $\mathcal{S}(|X_i|+\ldots+|X_{m/2}|+|Y|)$, $1 \leq i \leq m/2$, corresponds to the star with center v_i and leaves $X_i \cup \cdots \cup X_{m/2} \cup Y$. Similarly, term $\mathcal{S}(|Z_i|+\ldots+|Z_{m/2}|+|Y|)$, $1 \leq i \leq m/2$, corresponds to the star with center v_{m+2-i} and leaves $Z_i \cup \cdots \cup Z_{m/2} \cup Y$. Finally term $\mathcal{S}(|Y|)$ corresponds to the star with center $v_{m/2+1}$ and leaves Y, and $\mathcal{K}(|V|)$ corresponds to the clique V. By substituting the cardinalities of the sets, we obtain the bound of Lemma 4. □

We now show the following upper bound on $\mathcal{W}(\mathcal{H}(B,m))$.

Lemma 5. $\mathcal{W}(\mathcal{H}(B,m)) \leq 2\mathcal{K}(3m^3(B+1)+m(B+1)) + 2\sum_{i=1}^{m/2} \mathcal{S}(2(m-i+1)(B+1)) + \mathcal{S}(m(B+1)) - (B+1)\mathcal{K}(m+1)$.

Proof. Consider the following layout of $\mathcal{H}(B,m)$:

$$R_1, X_1, \cdots, X_{m/2}, v_1, Y_1, v_2, Y_2, \ldots, Y_m, v_{m+1}, Z_{m/2}, \cdots, Z_1, R_2, \tag{1}$$

where $Y_1 \cup \cdots \cup Y_m = Y$, and for each $1 \leq i \leq m$, $|Y_i| = B$. Observe that the order of vertices inside R_1, X_i, Y_i, Z_i, $1 \leq i \leq \frac{m}{2}$, and R_2 is irrelevant.

Since vertices in $R_1 \cup X$ and $Z \cup R_2$ are consecutive in the layout, the contribution of cliques $R_1 \cup X$ and $Z \cup R_2$ is respectively $\mathcal{K}(|R_1|+|X|) = \mathcal{K}(3m^3(B+1)+m(B+1))$ and $\mathcal{K}(|Z|+|R_2|) = \mathcal{K}(3m^3(B+1)+m(B+1))$.

Now consider vertices $v_1, \ldots, v_{m/2}$. Each vertex v_i, $1 \leq i \leq m/2$, has $2(m-i+1)(B+1)$ neighbors in graph $\mathcal{H}(B,m)$: $2(m/2-i+1)(B+1)$ neighbors belonging to $X_i, \ldots, X_{m/2}$, m neighbors $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{m+1}$, and mB neighbors in Y. Observe now that these $2(m-i+1)(B+1)$ neighbors of v_i appear in (1) at consecutive positions before and after v_i and moreover, v_i appears exactly in the middle of those vertices. This implies that the contribution of each star centered at v_i $1 \leq i \leq m/2$ is $\mathcal{S}(2(m-i+1)(B+1))$ and the overall contribution is $\sum_{i=1}^{m/2} \mathcal{S}(2(m-i+1)(B+1))$.

Symmetrically, the contribution of the stars centered at $v_{m/2+1}, \ldots, v_{m+1}$ is also $\sum_{i=1}^{m/2} \mathcal{S}(2(m-i+1)(B+1))$. By the same argument, the star with center $v_{m/2}+1$ and leaves $\{v_1, \ldots, v_{m/2}, v_{m/2+2}, \ldots, v_{m+1}\}$ contributes with $\mathcal{S}(m(B+1))$.

Observe that each edge with both endpoints in $\{v_1, \ldots, v_{m+1}\}$ has been counted twice. We therefore have to subtract $(B+1)\mathcal{K}(m+1)$ to take this into account.

By summing up all the terms, we obtain the lemma. □

To proceed, we need to estimate from above the difference between the upper (Lemma 5) and lower (Lemma 4) bounds. By straightforward arithmetics, one can establish that for any x and $y \leq x$, we have $\mathcal{S}(x) - \mathcal{S}(x-y) \leq xy$. Using this, the difference between the upper and lower bounds is

$$2\sum_{i=1}^{m/2}[\mathcal{S}(2(m-i+1)(B+1)) - \mathcal{S}(2(m-i+1)(B+1)-m)] + [\mathcal{S}(m(B+1)) - \mathcal{S}(mB)] - (B+2)\mathcal{K}(m+1) \leq 2\sum_{i=1}^{m/2} 2(m-i+1)(B+1)m + m^2(B+1) - (B+2)\mathcal{K}(m+1) \leq 4m(B+1)\sum_{i=1}^{m/2}(m-i+1) + m^2(B+1) - (B+2)m(m+1)(m+2)/6 < 3m^3(B+1) \qquad (2)$$

The next step is to prove that layout (1) of Lemma 5 is actually an optimal linear arrangement. Let L^* by an optimal linear arrangement of $\mathcal{H}(B,m)$. We first show that L^* maps vertices of $R_1 \cup X$ to consecutive positions.

Lemma 6. *Let L^* be an optimal linear arrangement of $\mathcal{H}(B,m)$. Then the set $\{L^*(w)| w \in R_1 \cup X\}$ contains $|R_1| + |X|$ consecutive integers.*

Proof. Assume for contradiction that some vertex from $V \cup Y \cup R_2$ appears at a position p which is between the smallest and the largest positions of $\{L^*(w)|w \in R_1 \cup X\}$. Then the contribution of each edge of $\{(w_1,w_2)|w_1,w_2 \in R_1 \cup X, L^*(w_1) < p, L^*(w_2) > p\}$ is increased by at least one. The total increase is then at least $\min_{1\leq L\leq |R_1|+|X|-1}(L \cdot (|R_1|+|X|-L)) = |R_1|+|X|-1 = 3m^3(B+1) + m(B+1) - 1$. Observe now that this quantity is larger than the maximal possible difference (2) between the upper and the lower bound on $\mathcal{W}(\mathcal{H}(B,m))$, which gives the desired contradiction. □

Lemma 7. *Let L^* be an optimal linear arrangement of $\mathcal{H}(B,m)$. Then the set $\{L^*(w)| w \in Z \cup R_2\}$ contains $|Z| + |R_2|$ consecutive integers.*

Proof. By symmetry, the proof is similar to that of Lemma 6. □

Thus, Lemmas 6 and 7 imply that any optimal linear arrangement maps vertices of $R_1 \cup X$ and $Z \cup R_2$ into sets of consecutive positions. By an argument similar to that of Lemma 6, we further deduce that vertices of $R_1 \cup X$ appear in the beginning of an optimal layout, and vertices of $Z \cup R_2$ appear in the end of this layout, while the other vertices ($V \cup Y$) appear between them. Indeed, if it is not the case, edges “crossing” $R_1 \cup X$ (or $Z \cup R_2$) would give an increase in the weight that would be larger than the maximal possible difference (2) between the upper and the lower bound.

To further specify an optimal linear arrangement of $\mathcal{H}(B,m)$, we have to clarify the layout of $V \cup Y$. The following lemma completes this part of the proof.

Lemma 8. *Any optimal linear arrangement of $\mathcal{H}(B,m)$ has the form*

$$R_1 \cup X, v_1, Y_1, v_2, Y_2, \ldots, Y_m, v_{m+1}, Z \cup R_2, \qquad (3)$$

where $Y_1 \cup \cdots \cup Y_m = Y$ and for each $1 \leq i \leq m$, $|Y_i| = B$.

Proof. It is easy to see that v_1 appears immediately after $R_1 \cup X$, as otherwise it can be moved down to that position which only decreases the resulting weight. By symmetry,

v_{m+1} appears immediately before $Z \cup R_2$. From similar considerations, we can deduce that the ordering of vertices in V is the "natural" ordering $v_1, v_2, \ldots, v_{m+1}$ (otherwise by permuting the vertices we would decrease the total weight).

It remains only to show that between each v_i and v_{i+1} there are exactly B vertices of Y. If this is the case, then observe (see the proof of Lemma 5) that each star centered at v_i has exactly the same number of neighbors to the left of $L^*(v_i)$ as to the right of $L^*(v_i)$, and all these neighbors appear at consecutive positions. Thus, each star centered at v_i is optimally arranged and reaches the absolute lower bound of the contributed weight. Any other arrangement of $v_1, \ldots, v_{m+1}$ would break the parity at least for one of these stars, and therefore, by the remark after Lemma 2, would necessarily increase the weight contributed by this star. This completes the proof. □

3.2 NP-Completeness Proof

Using the construction of graph $\mathcal{H}(B, m)$ from the previous section, we now prove Theorem 1 by reduction from the 3-PARTITION.

Consider an instance of 3-PARTITION, $(\{a_1, \ldots, a_{3m}\}, B)$, where $\sum_{i=1}^{3m} a_i = mB$. We transform it into the graph $\mathcal{H}(B, m)$ extended by additional edges over vertices in Y. Consider a partition $Y = Y_1 \cup \cdots \cup Y_{3m}$, where $Y_i \cap Y_j = \emptyset$ for $i \neq j$, and $|Y_i| = a_i$ for all i, $1 \leq i \leq 3m$. We turn each Y_i into a clique by adding a set of edges E_i over all pairs of vertices of Y_i. Consider an extended graph $G = \mathcal{H}(B, m) \bigcup \cup_{i=1}^{3m} (Y_i, E_i)$. Again, from Figure 1, it is clear that G is an interval graph. Let $K = \mathcal{W}(\mathcal{H}(B, m)) + \sum_{i=1}^{3m} \mathcal{K}(a_i)$. Since the time running of this transformation depends on B, the whole transformation can be carried out in polynomial time.

Theorem 2. *There exists a 3-partition of* $\{a_1, \ldots, a_{3m}\}$ *if and only if* $\mathcal{W}(G) = K$.

Proof. **Only if part:** Assume that $A = \{a_1, \ldots, a_{3m}\}$ can be partitioned into m disjoint subsets $A_1, \cdots, A_m$, each summing up to B. Let $A_i = \{a_1^i, \ldots, a_{|A_i|}^i\} \subseteq A$. We construct a layout L^* defined by

$$R_1 \cup X, v_1, Y_1^1, \ldots, Y_{|A_i|}^1, v_2, \ldots, Y_1^m, \ldots, Y_{|A_m|}^m, v_{m+1}, Z \cup R_2, \tag{4}$$

where $Y_j^i \in \{Y_1, \ldots, Y_{3m}\}$ is the subset corresponding to a_j^i ($|Y_j^i| = a_j^i$). Observe that in (4), there are exactly B vertices of Y between every v_i and v_{i+1} and that all edges between vertices of Y are edges of cliques with vertices mapped by L^* to consecutive positions. Therefore, using Lemma 8, the weight of L^* is $\mathcal{W}(G, L^*) = \mathcal{W}(\mathcal{H}(B, m)) + \sum_{i=1}^{3m} \mathcal{K}(a_i) = K$. By Corollary 1, this is the smallest possible weight, i.e., $\mathcal{W}(G) = K$.

If part: Let $\mathcal{W}(G) = K$, i.e., there exists a layout L^* such that $\mathcal{W}(G, L^*) = K$. Decompose G as the edge-disjoint union of graph $\mathcal{H}(B, m)$ and cliques $(Y_1, E_1), \ldots, (Y_{3m}, E_{3m})$. For any layout L of G, $\mathcal{W}(\mathcal{H}(B, m), L) \geq \mathcal{W}(\mathcal{H}(B, m))$ and $\mathcal{W}((Y_i, E_i), L) \geq \mathcal{K}(a_i)$ for all i, $1 \leq i \leq 3m$. On the other hand, by Corollary 1, $\mathcal{W}(G) \geq \mathcal{W}(\mathcal{H}(B, m)) + \sum_{i=1}^{3m} \mathcal{K}(a_i)$. Therefore, if a layout L^* verifies $\mathcal{W}(G, L^*) = K$, this implies that *(i)* $\mathcal{W}(\mathcal{H}(B, m), L^*) = \mathcal{W}(\mathcal{H}(B, m))$ and *(ii)* $\mathcal{W}((Y_i, E_i), L^*) = \mathcal{K}(a_i)$, for all i, $1 \leq i \leq 3m$.

Condition *(i)* implies that layout L^* verifies Lemma 8, and, in particular, splits vertices of Y by vertices $v_1, \ldots, v_{m+1}$ into m groups, each of cardinality B. Condition *(ii)* ensures that each subset Y_i is mapped by L^* into consecutive positions and therefore falls inside one such group. This means that numbers $\{a_1, \ldots, a_{3m}\}$ (cardinalities of $\{Y_1, \ldots, Y_{3m}\}$) are split into m disjoint subsets each of which sums up to B. This completes the proof of Theorem 2. □

Since the optimal linear arrangement problem for interval graphs is NP-complete, the next section describes a 2-approximation algorithm for interval graphs.

4 A 2-Approximation Algorithm for OLA of Interval Graphs

Before describing an approximation algorithm, we study two layouts of an interval graph G, defined by any fixed interval model. Let $\mathcal{I}$ be an interval model of G. The layout of G consisting of vertices ordered by the left endpoints of their corresponding intervals is called the *left endpoint ordering* (*leo*) of G with respect to the interval model $\mathcal{I}$. Similarly, the layout of G consisting of vertices ordered by the right endpoints of their corresponding intervals is called the *right endpoint ordering* (*reo*) of G with respect to $\mathcal{I}$.

It has been shown in [15] that leo and reo are good approximations for the bandwidth of interval graphs: $b(G, leo) \leq 2 \cdot bw(G)$ and $b(G, reo) \leq 2 \cdot bw(G)$. This is based on the fact that:

- in a left endpoint ordering, *leo*, for every pair of adjacent vertices $leo(u) < leo(w)$, each vertex between u and w is adjacent to u, and
- in a right endpoint ordering *reo*, for every pair of adjacent vertices $reo(u) < reo(w)$ each vertex between u and w is adjacent to w.

This can be used to show that left endpoint and right endpoint orderings are 2-approximations for the OLA problem on interval graphs.

Theorem 3. *Let $G = (V, E)$ be an interval graph, and let $\mathcal{I}$ be an interval model of G. Then, $\mathcal{W}(G, leo) \leq 2\mathcal{W}(G)$, and $\mathcal{W}(G, reo) \leq 2\mathcal{W}(G)$.*

Proof. We focus on the ordering reo. For any integer i, $1 \leq i \leq V(G)$, we define graph G_i such that

- $V(G_i) = \{u \mid u \in V(G) \land reo(u) \leq i\}$, and
- $E(G_i) = \{e = (u, v) \in E(G) \mid u \in V(G_i) \land v \in V(G_i)\}$.

We prove this theorem by induction on the number of vertices. The induction hypothesis is that $\mathcal{W}(G_i, reo) \leq 2\mathcal{W}(G_i)$ for any integer i , $1 \leq i \leq V(G)$.

The basis of the induction is the situation where G_1 contains only one vertex ($i = 1$). The induction hypothesis holds here because $\mathcal{W}(G_1, reo) = 0$ and $\mathcal{W}(G_1) = 0$. Then $\mathcal{W}(G_1, reo) \leq 2\mathcal{W}(G_1)$.

For the induction step, we assume that the induction hypothesis for i holds. Now, we will prove that the induction hypothesis holds for $i + 1$. Let u be the vertex such that $reo(u) = i + 1$.

First we give a lower bound for $\mathcal{W}(G_{i+1})$. We can notice that sets $E(G_i)$ and $\{e = (v, u) \mid v \in V(G_i) \land e \in E(G_{i+1})\}$ form a partition of set $E(G_{i+1})$. By Lemma 3, $\mathcal{W}(G_{i+1}) \geq \mathcal{W}(G_i) + \mathcal{W}(S_{d_{G_{i+1}}(u)})$, Secondly, we give an upper bound for $\mathcal{W}(G_{i+1}, reo)$ by considering the partition $E(G_i)$ and $\{e = (v,u) \mid v \in V(G_i) \wedge e \in E(G_{i+1})\}$ of set $E(G_{i+1})$.

For the edge set $E(G_i)$, we have $\sum_{e=(u,v)\in E(G_i)} |reo(u) - reo(v)| = \mathcal{W}(G_i, reo)$.

For the edge set $\{e = (v,u) \mid v \in V(G_i) \wedge e \in E(G_{i+1})\}$, since vertex u and its neighborhood in G_{i+1} are consecutive in the layout reo, the linear arrangement reo gives $d_{G_{i+1}}(u) + 1$ consecutive numbers. We can compute an upper bound of $\sum_{v\in N_{G_{i+1}}(u)} |reo(u) - reo(v)|$ because according to the linear arrangement reo, we are in the situation of the worst case for the star. So, we have

$$\sum_{v\in N_{G_{i+1}}(u)} |reo(u) - reo(v)| \leq 2\mathcal{W}(S_{d_{G_{i+1}}(u)})$$

This yields an upper bound for $\mathcal{W}(G_{i+1}, reo)$. We get $\mathcal{W}(G_{i+1}, reo) \leq \mathcal{W}(G_i, reo) + 2\mathcal{W}(S_{d_{G_{i+1}}(u)})$.

Since we have $\mathcal{W}(G_i, reo) \leq 2\mathcal{W}(G_i)$ by induction hypothesis, we have

$$\mathcal{W}(G_{i+1}, reo) \leq 2\mathcal{W}(G_i) + 2\mathcal{W}(S_{d_{G_{i+1}}(u)}) \leq 2\mathcal{W}(G_{i+1})$$

So, the Theorem holds. □

Theorem 3 shows that left endpoint and right endpoint orderings are 2-approximation algorithms for this problem. This is the best possible bound for these orderings. In fact, a star S_α with an even number α of leaves has an interval representation such that $\mathcal{W}(S_\alpha, reo) = \frac{\alpha(\alpha+1)}{2}$ and $\mathcal{W}(S_\alpha) = \frac{\alpha}{2}(\frac{\alpha}{2} + 1)$. So the ratio $\frac{\mathcal{W}(S_\alpha, reo)}{\mathcal{W}(S_\alpha)}$ equals to $2 - \frac{1}{\alpha+2}$.

In the next section, we focus on close relatives of interval graphs – permutation graphs – and on their generalization – cocomparability graphs.

5 OLA of Permutation and Cocomparability Graphs

Cocomparability, interval, and permutation graphs are well-known classes of perfect graphs. All of them have geometric intersection models. Many references, including [2,12], contain comprehensive overviews of the many known structural and algorithmic properties of (co)comparability, interval, and permutation graphs.

Permutation graphs are intersection graphs of straight line segments between two parallel lines. Vertices of the graph are associated to segments and two vertices are adjacent iff corresponding segments intersect.

Our first remark here is that graph $\mathcal{H}(B, m)$ considered in Section 3 is a permutation graph. Figure 2 shows a permutation representation for $\mathcal{H}(B, m)$.

This immediately implies

Lemma 9. *The problem of deciding, for a permutation graph $G = (E, V)$ and a constant K, whether $\mathcal{W}(G) \leq K$ is NP-complete.*

Let us now turn to cocomparability graphs that are generalizations of both interval and permutation graphs. A graph G is cocomparability if its complement $\overline{G}$ is a comparability graph, i.e., the comparability graph of a poset $P = (V, \prec)$ is the graph with vertex set V for which vertices x and y are adjacent if and only if either $x \prec y$ or $y \prec x$ in P.

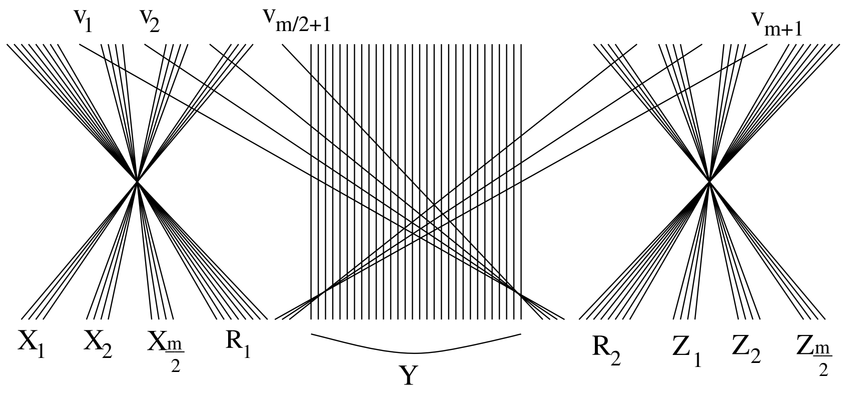


Fig. 2. Permutation representation of graph $\mathcal{H}(B, m)$

The following property of cocomparability graphs is well known (see e.g. [2]), and it is crucial for our arguments.

Proposition 1. *A graph $G = (V, E)$ is a cocomparability graph if and only if it has a* cocomparability ordering*, i.e., an ordering $(v_1, v_2, \ldots, v_n)$ of its vertices such that $(v_i, v_k) \in E$ and $i < j < k$ imply either $(v_i, v_j) \in E$ or $(v_j, v_k) \in E$.*

Since every interval graph is a cocomparability graph, the OLA problem remains NP-complete on cocomparability graphs. Now, we focus on the approximation problem. First, the following lower bound for the weight of an optimal linear arrangement of any graph will be useful when analyzing the performance ratio of some algorithms and orderings respectively.

Lemma 10. *For every graph $G = (V, E)$,*

$$\mathcal{W}(G) \geq \frac{m}{2} + \frac{1}{8}\sum_{v \in V} d^2(v).$$

Proof. Let v be a vertex of G. Then to minimize the sum over all edges incident to v in a layout, half of the neighbors of v must be placed immediately to the left of v and half of the neighbors of v must be placed immediately to the right of v. Thus the sum over all edges incident to v is at least $1 + 1 + 2 + 2 + \cdots + \frac{d(v)}{2} + \frac{d(v)}{2}$ if $d(v)$ is even, and $1 + 1 + 2 + 2 + \cdots + \frac{d(v)-1}{2} + \frac{d(v)-1}{2} + \frac{d(v)+1}{2}$ if $d(v)$ is odd.

Thus we obtain

$$\mathcal{W}(G) \geq \frac{1}{2}\sum_{v \in V}\Big(\big(\frac{d(v)}{2} + 1\big)\big(\frac{d(v)}{2}\big)\Big) \geq \sum_{v \in V}\big(\frac{d^2(v)}{8} + \frac{d(v)}{4}\big)$$
$$\geq \frac{1}{8}\sum_{v \in V} d^2(v) + \frac{m}{2}. \qquad \square$$

We use the lower bound of the previous section to show that every cocomparability ordering of a cocomparability graph has weight at most $8 \cdot \mathcal{W}(G)$.

Theorem 4. *Let $G = (V, E)$ be a cocomparability graph and let L be a cocomparability ordering of G. Then, $\mathcal{W}(G, L) \leq 8 \cdot \mathcal{W}(G)$.*

Proof. By the definition of L, if u and w are adjacent in G then all vertices between u and w in L are either adjacent to u or adjacent to w. Therefore

$$|L(u) - L(w)| \leq |N(u) \cup N(w)| \leq d(u) + d(v),$$

and by Lemma 10,

$$\begin{aligned}\mathcal{W}(G, L) &= \sum_{e=(u,v)} |L(u) - L(v)| \\ &\leq \sum_{e=(u,v)} (d(u) + d(v)) \\ &\leq \sum_{v \in V} d^2(v) \\ &\leq 8 \cdot \mathcal{W}(G).\end{aligned}$$

□

Since a cocomparability ordering can be found in polynomial time $O(n^{2.376})$ [16], Theorem 4 immediately implies an 8-approximation polynomial-time algorithm for OLA on cocomparability graphs.

6 Conclusion and Open Problems

In this paper, we resolved the complexity of the OLA problem for interval, permutation and consequently for cocomparability, graphs. We have given simple approximation algorithms for those classes. There are several other linear layout problems, like CUTWIDTH, whose complexity is not resolved for the class of interval graphs [4].

References

1. T. Biedl, B. Brejova, E. Demaine, A. Hamel, A. Lopez-Ortiz, T. Vinar, Finding Hidden Independent Sets in Interval Graphs, *Theoretical Computer Science* 310 (1-3), Jan 2004, 287–307.
2. A. Brandstädt, V.B. Le, J. Spinrad Graph Classes: A Survey, *SIAM Monographs on Discrete Math. Appl., Vol.* 3, SIAM, Philadelphia (1999)
3. F.R.K. Chung, Labelings of graphs, *Selected Topics in graph theory*, 151-168, Academic Press, San Diego, 1988.
4. J. Díaz, J. Petit, M.J. Serna, A survey of graph layout problems, *ACM Computing Surveys*, vol.34, No.3, Sept 2002, 313-356.
5. G. Even, S. Naor, B.Schieber, S. Rao. G. Even, Divide-and-conquer approximation algorithms via spreading metrics, *Journal of the ACM*, 47(4):585–616, 2000.
6. S. Even, Y. Shiloach, NP-Completeness of Several Arrangement Problems, *Technical Report #43*, Computer Science Dept., The Technion, Haifa, Israel, 1975.

7. M.A. Goldberg, I.A. Klipker, Minimal placing of trees on a line, *Technical report, Physico-Technical Institute of Low Temperatures, Academy of Sciences of Ukranian SSR, USSR*, 1976.
8. M. Farach-Colton, Y. Huang, J.L.L. Woolford, Discovering temporal relations in molecular pathways using protein-protein interactions, *Proceedings of the 8th Annual International Conference on Computational Molecular Biology (RECOMB)*, 2004, San Diego, California, USA, March 27-31, 2004, ACM Press, 150–156.
9. U. Feige, Approximating the bandwidth via volume respecting embeddings, *J. Comput. System Sci.* **60** (2000) 510-539.
10. M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth, Complexity results for bandwidth minimization, *SIAM J. Appl. Math.*, **34** (1978), 477–495.
11. M.R. Garey, D.S. Johnson, Computers and Intractability - A Guide to the Theory and Practice of NP-Completeness, W.H. Freeman, San Francisco, 1979
12. M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, *Academic Press, New York* (1980)
13. D. Johnson, The NP-completeness column: an ongoing guide, *J. Algorithms* **6** (1985), 434–451.
14. D.J. Kleitman, R.V. Vohra, Computing the bandwidth of interval graphs, *SIAM J. Discrete Math.* **3** (1990) 373-375.
15. D. Kratsch, L.K. Stewart, Approximating bandwidth by mixing layouts of interval graphs, *SIAM Journal on Discrete Mathematics* **15** (2002) 435-449.
16. R. McConnell, J. Spinrad, Modular decomposition and transitive orientation, *Discrete Math.* **201** (1999), 189–241.
17. B. Monien, The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. Algebraic Discrete Methods* **7** (1986), 505–512.
18. S. Rao, A. Richa, New approximation techniques for some ordering problems, *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98)* 211–218, ACM, New York, 1998.
19. Y. Shiloach, A minimum linear arrangement algorithm for undirected trees, *SIAM Journal on Computing* **8** (1979) 15-32.
20. A.P. Sprague, An $O(n \log n)$ algorithm for bandwidth of interval graphs, *SIAM J. Discrete Math.* **7** (1994) 213-220
21. W. Unger, The complexity of the approximation of the bandwidth problem, *Proceedings of the Thirty-ninth Annual IEEE Symposium on Foundations of Computer Science* (Palo Alto, CA, 1998).
22. M. Waterman, Introduction to computational biology: Maps, sequences and genomes, Chapman & Hall, 1995.

The Lempel-Ziv Complexity of Fixed Points of Morphisms

Sorin Constantinescu and Lucian Ilie[⋆,⋆⋆]

Department of Computer Science, University of Western Ontario
London, Ontario, N6A 5B7, Canada
{sorinco, ilie}@csd.uwo.ca

Abstract. The Lempel–Ziv complexity is a fundamental measure of complexity for words, closely connected with the famous LZ77, LZ78 compression algorithms. We investigate this complexity measure for one of the most important families of infinite words in combinatorics, namely the fixed points of morphisms. We give a complete characterisation of the complexity classes which are $\Theta(1)$, $\Theta(\log n)$, and $\Theta(n^{1/k})$, $k \in \mathbb{N}$, $k \geq 2$, depending on the periodicity of the word and the growth function of the morphism. The relation with the well-known classification of Ehrenfeucht, Lee, Rozenberg, and Pansiot for factor complexity classes is also investigated. The two measures complete each other, giving an improved picture for the complexity of these infinite words.

1 Introduction

Before publishing their famous papers introducing the well-known compression schemes LZ77 and LZ78 in [31] and [32], resp., Lempel and Ziv introduced a complexity measure for words in [18] which attempted to detect "sufficiently random looking" sequences. In contrast with the fundamental measures of Kolmogorov [16] and Chaitin [3], Lempel and Ziv's measure is computable. The definition is purely combinatorial; its basic idea, splitting the word into minimal never seen before factors, proved to be at the core of the well-known compression algorithms, as well as subsequent variations. Another, closely related, variant is to decompose the word into maximal already seen factors.

Lempel–Ziv-type complexity and factorisations have important applications in many areas, such as, data compression [31,32], string algorithms [6,22,17,27], molecular biology [14,13,4], and neural computing [29,1,30].

Lempel and Ziv [18] investigate various properties which are expected from a complexity measure which intends to approach randomness. They prove it to be subadditive and also that most sequences are complex but still not too many; see [18] for details. Also, they test it against the de Bruijn words, [2], an well-established case of complex words – de Bruijn words contain as factors all words of a given length, within the minimum possible space. Therefore, they establish the first connection with the factor complexity, which is also one of our topics.

[⋆] Corresponding author.

[⋆⋆] Research partially supported by NSERC.

In this paper, we investigate the Lempel–Ziv complexity for one of the most important classes of infinite words in combinatorics, namely the fixed points of morphisms. Many famous infinite words, such as Fibonacci or Thue-Morse, belong to this family; see, e.g., [20].

It is due to the fundamental nature of this complexity measure that it gives a complete characterisation of the possible classes of complexity that may appear. The lowest complexity, constant, or $\Theta(1)$, is encountered for the simplest words, that is, ultimately periodic. For non-periodic words, the complexity depends on the growth function of the underlying morphism for the letter on which the morphism is iterated. Thus, for polynomial growth we obtain $\Theta(n^{1/k})$, $k \in \mathbb{N}$, $k \geq 2$, whereas for the exponential growth the complexity is $\Theta(\log n)$. We give examples for which each of the above complexities is reached.

Our results are similar with the well-known ones of Ehrenfeucht, Lee, and Rozenberg [7], Ehrenfeuct and Rozenberg [8,9,10,11,12,25] and Pansiot [23,24] who provided the same characterisation for the factor complexity. Comparing the two characterisations, we find out that they complete each other in an interesting way. While theirs distinguishes four complexities classes for the exponential case, ours gives an infinite hierarchy (given by the parameter k above) in the polynomial case, corresponding to Pansiot's quadratic complexity.

The paper is structured as follows. After some basic definitions in the next section, we introduce the Lempel–Ziv complexity and related concepts in Section 3. Section 4 contains an important intermediate result which characterises the complexity of powers of a morphism. Using it, our complete characterisation is proved in Section 5 where examples which reach each complexity involved are shown. The comparison with Pansiot characterisation of factor complexity is included in Section 6. Some proofs had to be omitted due to limited space.

2 Basic Notions

We introduce here the basic definitions and concepts we need. For further details we refer the reader to [19,20,21,5].

Let Σ be an alphabet (finite non-empty set) and denote by Σ^* the free monoid generated by Σ, that is, the set of all finite words over Σ. The elements of Σ are called letters and the empty word is denoted ε. The length of a word w is denoted $|w|$ and represents the number of letters in w; e.g., $|\mathsf{abaab}| = 5$ and $|\varepsilon| = 0$.

Given the words $w, x, y, z \in \Sigma^*$ such that $w = xyz$, x is called a *prefix*, y is a *factor* and z a *suffix* of w; we use the notation $x \leq w$. If moreover $x \neq w$, then x is a *proper* prefix of w, denoted $x < w$. The prefix of length n of w is denoted $\mathsf{pref}_n(w)$.

An infinite word is a function $w : \mathbb{N} - \{0\} \to \Sigma$. A finite word can be viewed as a function $w : \{1, 2, \ldots, |w|\} \to \Sigma$. In either case, the factor of w starting at position i and ending at position j, will be denoted by $w(i, j) = w_i w_{i+1} \ldots w_j$. The set of all factors of w is $F(w)$. The set of letters of Σ that actually occur in w is denoted $\Sigma(w)$. The set of infinite words over Σ is denoted Σ^ω. An infinite word w is *ultimately periodic* if $w = uvvv\ldots$, for some $u, v \in \Sigma^*$, $v \neq \varepsilon$. When we say w is non-periodic, we mean it is not ultimately periodic.

A morphism is a function $h : \Sigma^* \to \Delta^*$ such that $h(\varepsilon) = \varepsilon$ and $h(uv) = h(u)h(v)$, for all $u, v \in \Sigma^*$. Clearly, a morphism is completely defined by the images of the letters in the domain. For all our morphisms, $\Sigma = \Delta$.

A morphism $h : \Sigma^* \to \Sigma^*$ is called *non-erasing* if $h(a) \neq \varepsilon$, for all $a \in \Sigma$, *uniform* if $|h(a)| = |h(b)|$, for all $a, b \in \Sigma$, and *prolongeable* on $a \in \Sigma$ if $a < h(a)$.

If h is non-erasing and prolongeable on a, then $h^n(a)$ is a proper prefix of $h^{n+1}(a)$, for all $n \in \mathbb{N}$. We say that the morphism h has a *fixed point* $w \in \Sigma^\omega$ given by

$$w = \lim_{n\to\infty} h^n(a) = h^\infty(a).$$

3 Word Histories and Lempel–Ziv Complexity

Let w be a (possibly infinite) word. We define the operator π that removes the final letter of a finite word w: $\pi(w) = w(1, |w| - 1)$.

We now introduce a fundamental notion of the Lempel–Ziv complexity: a *history* $H = (u_1, u_2, \ldots, u_n)$ of $w \neq \varepsilon$ is a factorisation of w, $w = u_1u_2\ldots u_n$, having the property that $u_1 \in \Sigma$ and

$$\pi(u_i) \in F(\pi^2(u_1u_2...u_i)), \text{ for all } 2 \leq i \leq n\ .$$

This definition requires that any new factor u_i, excepting its last letter, appears before in the word. However, it is still possible that the whole u_i does occur before in w, or $u_i \in F(\pi(u_1u_2...u_i))$. In this case u_i is called *reproductive*. Otherwise u_i is *innovative*.

Example 1. Consider the word $w = \mathsf{aaabaabbaba}$. A possible history of w is $(\mathsf{a}, \mathsf{aab}, \mathsf{aab}, \mathsf{bab}, \mathsf{a})$. The second and fourth components are innovative whereas the third and fifth are reproductive.

By definition, n is called the length of the history H and is denoted by $|H|$.

Two kinds of history are important to us. The first, directly connected to the definition of Lempel–Ziv complexity is the *exhaustive* history. A history H is exhaustive if all u_i, $2 \leq i \leq |H| - 1$, are innovative. In other words, the whole new factor u_i does not occur before in the word even if all its proper prefixes do. Clearly, the exhaustive history of a word is unique.

By contrast with the exhaustive history, a *reproductive history* requires that all its factors have occurred before (they are reproductive), with the necessary exceptions of never seen before letters: a history $H = (u_1, u_2, \ldots, u_n)$ is reproductive if either

(i) $u_i \in F(\pi(u_1u_2...u_i))$ or
(ii) $u_i \notin F(\pi(u_1u_2...u_i))$ but then $u_i \in \Sigma$.

The innovative factors in a reproductive history are single letters. A reproductive history need not be unique.

Example 2. For the word in Example 1, $(\mathsf{a}, \mathsf{aab}, \mathsf{aabb}, \mathsf{aba})$ is the exhaustive history, whereas $(\mathsf{a}, \mathsf{aa}, \mathsf{b}, \mathsf{aa}, \mathsf{b}, \mathsf{ba}, \mathsf{ba})$ and $(\mathsf{a}, \mathsf{aa}, \mathsf{b}, \mathsf{aab}, \mathsf{ba}, \mathsf{ba})$ are two reproductive histories.

The following result, due to [18], relates the exhaustive history with all other histories of a word.

Lemma 1. *The exhaustive history of a word is the shortest history of that word.*

By definition, the *Lempel–Ziv complexity* of a finite word w, denoted $\textsc{lz}(w)$, is the length of the exhaustive history of w. Therefore, by Lemma 1, for any history H, $\textsc{lz}(w) \le |H|$. This extends naturally to infinite words as follows: the *Lempel–Ziv complexity of an infinite word* w is the function $\textsc{lz}_w : \mathbb{N} \to \mathbb{N}$, defined by

$$\textsc{lz}_w(n) = \textsc{lz}(\mathsf{pref}_n(w)), \text{ for all } n \ge 0 ,$$

as the complexity of finite prefixes of w.

4 The Complexity of Powers

The main result of this section is that the complexity of the powers of h applied to the generator of its fixed point, a, is either linear or bounded, that is, either $\textsc{lz}(h^n(a)) = \Theta(n)$ or $\textsc{lz}(h^n(a)) = \Theta(1)$.

Given the morphism h, we can assume, without loss of generality, that each letter of Σ occurs in $h^\infty(a)$, the fixed point of h. If that is not the case, h can be restricted to the set of those letters that do occur in w and the fixed point of the restriction will still be the same.

We show first that the complexity of powers is at most linear. To this end, we introduce the *maximal reproductive history* of a finite word w, denoted $RH(w)$. For $w = w_1 w_2 \dots w_{|w|}$, $w_i \in \Sigma$, we define

$$RH(w) = (u_1, u_2, \dots, u_n)$$

as follows:

– $u_1 = w_1$, the first letter of w,

– $u_{i+1} = \begin{cases} w_{|u_1u_2\dots u_i|+1}, & \text{if } w_{|u_1u_2\dots u_i|+1} \notin \Sigma(u_1u_2\dots u_i) \\ \text{longest } w \text{ with } w \in F(\pi(u_1u_2\dots u_i w)), & \\ & \text{if } w_{|u_1u_2\dots u_i|+1} \in \Sigma(u_1u_2\dots u_i) \end{cases}$

for all $i \ge 2$.

With the exception of new single letters, $RH(w)$ is created by taking at each step the maximal factor that has occurred before. For the word in Example 1, the maximal reproductive history is $(\mathsf{a}, \mathsf{aa}, \mathsf{b}, \mathsf{aab}, \mathsf{ba}, \mathsf{ba})$.

From the definition it is clear that $RH(w)$ is a reproductive history. It follows from Lemma 1 that $|RH(w)| \ge \textsc{lz}(w)$.

Remark 1. The maximal reproductive history seems more natural than the exhaustive history. Indeed, all applications we mentioned above use the former factorisation. On the other hand, they are very closely related. For historical reasons, we defined the Lempel–Ziv complexity as the number of factors in the exhaustive history (as in [18]) but our results hold as well for the maximal reproductive history. This can be seen directly, by looking at the proofs, or from the following lemma which connects the lengths of the two histories.

Lemma 2. *For any $w \in \Sigma^*$, we have*

$$\textsc{lz}(w) \le |RH(w)| \le 2\,\textsc{lz}(w) - 1 .$$

The next step is to iterate reproductive histories through a morphism h. We will show a way to create a reproductive history of $h(w)$, given a reproductive history of w.

Let w be a word and $H = (v_0, v_1, \ldots, v_n)$ a reproductive history of w. Let $1 = i_1 < i_2 < \ldots < i_{|\Sigma(w)|}$ be the indexes corresponding to the single letter factors of H that have not occurred before. We define a factorisation of $h(w)$, denoted $h(H)$, by replacing all factors of w that have occurred before by their image through h and the single letters v_{i_j}, by the history $RH(h\left(v_{i_j}\right))$. We claim that this is a reproductive history of $h(w)$.

Example 3. Let us consider the Thue–Morse morphism

$$t(\mathsf{a}) = \mathsf{ab}, \quad t(\mathsf{b}) = \mathsf{ba},$$

and the word from Example 1, $w = \mathsf{aaabaabbaba}$. A reproductive history H (in fact, $RH(w)$) and its image through t, $t(H)$, are:

$$\begin{array}{rcl} H & = & (\mathsf{a}\ ,\quad \mathsf{aa}\ ,\quad \mathsf{b}\ ,\quad \mathsf{aab}\ ,\quad \mathsf{ba}\ ,\quad \mathsf{ba})\ , \\ h(H) & = & (\mathsf{a}, \mathsf{b},\ \ \mathsf{abab},\ \ \mathsf{b}, \mathsf{a},\ \ \mathsf{ababba},\ \ \mathsf{baab},\ \ \mathsf{baab})\ . \end{array}$$

Lemma 3. *If H is a reproductive history of w, then $h(H)$ is a reproductive history of $h(w)$.*

With respect to the length of $h(H)$, we note that each letter in $\Sigma(w)$, originally a stand alone factor of H, is transformed into the factorisation $RH(h\left(v_{i_j}\right))$ and, consequently, each letter x of w prompts a $|RH(x)| - 1$ increase in the length of $h(H)$:

$$|h(H)| \le |H| + \sum_{x \in \Sigma(w)} (|RH(x)| - 1)\ .$$

If we assume that all letters of Σ occur in w, then the increase in length is constant, which leads us to the following result.

Proposition 1. *If $h : \Sigma^* \to \Sigma^*$ is non-erasing and $a < h(a)$, $a \in \Sigma$, then $\textsc{lz}(h^n(a)) = O(n)$.*

The next result gives the inferior asymptotic limit for $\textsc{lz}(h^n(a))$. It is obvious that $\textsc{lz}(h^n(a))$, as a function of n, is increasing since $h^n(a) < h^{n+1}(a)$. The remaining part of this section is dedicated to showing that the growth of the Lempel–Ziv complexity of powers is at least linear unless the fixed point word is ultimately periodic.

Throughout the rest of this section, h is assumed to be a non-erasing morphism whose fixed point $w = h^\infty(a)$ is of the form

$$a\,u\,h(u)\dots h^n(u)h^{n+1}(u)\dots$$

where $h(a) = au$, $u \in \Sigma^*$, $u \neq \varepsilon$.

The following technical result is very important for the proof of the main result of this section.

Lemma 4. *If $h^q(u)h^{q+1}(u)h^{q+2}(u)$ occurs before its last occurrence in*

$$h^{q+3}(a) = a\,u\,h(u)\dots h^q(u)h^{q+1}(u)h^{q+2}(u)$$

and $|h^q(u)| < |h^{q+1}(u)|$, then $h^\infty(a)$ is ultimately periodic.

In order to be able to use Lemma 4, we need to find values of q for which $|h^q(u)| < |h^{q+1}(u)|$. It is clear that $|h^q(u)| \leq |h^{q+1}(u)|$ and, if there exists a letter z in $h^q(u)$ satisfying $|h(z)| \geq 2$, the inequality is strict. The following lemma says that such powers must exist or else the fixed point is ultimately periodic.

Lemma 5. *If $h(a) = au$, $u \in \Sigma^*$, $u \neq \varepsilon$, then there exist $m, p \in \mathbb{N}$ such that $|h^{m+jp}| < |h^{m+jp+1}|$, for all $j \geq 0$, or else $h^\infty(a)$ is ultimately periodic.*

Using Lemmata 4 and 5, we obtain, for all $j \geq 0$, that either

$$h^{m+jp}(u)h^{m+jp+1}(u)h^{m+jp+2}(u) \tag{1}$$

has never occurred before or w is ultimately periodic.

If we assume $w = h^\infty(a)$ to be non-periodic, then all factors of the form (1) can never occur before their last occurrence. This shows that there must exist a factor in the exhaustive history of w that ends within each distinct factor of the above mentioned form. It follows that

$$\textsc{lz}(h^n(a)) \geq \frac{1}{k}(n - n_0) + \textsc{lz}(h^{n_0+1}(a))$$

or $\textsc{lz}(h^n(a)) = \Omega(n)$. We proved

Proposition 2. *Exactly one of the following is true:*

1. *$w = h^\infty(a)$ is ultimately periodic,*
2. *$\textsc{lz}(h^n(a)) = \Omega(n)$.*

We can now enounce the main theorem of this section.

Theorem 1. *For a non-erasing morphism h that admits the fixed point $h^\infty(a)$, $\textsc{lz}(h^n(a))$ is either $\Theta(1)$ if $h^\infty(a)$ is ultimately periodic or $\Theta(n)$ otherwise.*

Proof. The fact that ultimate periodicity is equivalent to a bounded Lempel–Ziv complexity has been mentioned in [15]. The other case is a direct consequence of the above results. □

5 Growth Functions and Infinite Word Complexity

Let w be an infinite word generated by a non-erasing morphism h, $w = h^{\infty}(a)$. The prefix of a given length m of w will fall between two consecutive powers of h:

$$h^{n(m)}(a) \le \mathsf{pref}_m(w) < h^{n(m)+1}(a) \tag{2}$$

for a $n(m) \in \mathbb{N}$. If $\textsc{lz}(h^n(a))$ is bounded then $\textsc{lz}_w(n)$ is bounded. This establishes our first case for the complexity of $\textsc{lz}_w(\cdot)$, $\Theta(1)$.

For the case when $\textsc{lz}(h^n(a))$ is not bounded, we need more definitions and results. The *growth function* of the letter $x \in \Sigma$ in h is the function $h_x : \mathbb{N} \to \mathbb{N}$, defined by

$$h_x(n) = |h^n(x)|, \text{ for all } n \ge 0 \;.$$

The following result from [28,26] is very useful.

Lemma 6. *There exist an integer $e_a \ge 0$ and an algebraic real number $\rho_a \ge 1$ such that*

$$h_a(n) = \Theta(n^{e_a}\rho_a^n).$$

The pair (e_a, ρ_a) is called the *growth index* of a in h. We say that h_a (and a as well) is called:

- *bounded* if a's growth index w.r.t. h is $(0, 1)$,
- *polynomial*, if a's growth index w.r.t. h is $(e_a, 1)$, $e_a > 0$, and
- *exponential* if a's growth index w.r.t. h is (e_a, ρ_a), $e_a \ge 0$, $\rho_a > 1$.

Going back to our reasoning, since $\textsc{lz}(h^n(a))$ is not bounded, it has to be linear, by Theorem 1. Then a is not bounded and hence, by Lemma 6, we distinguish two cases:

1. $\rho_a = 1$ (h_a is polynomial). Then $|h^n(a)| = \Theta(n^{e_a})$ or $n(m) = \Theta(m^{1/e_a})$. Since, by (2),

$$\textsc{lz}(h^{n(m)}(a)) \le \textsc{lz}(\mathsf{pref}_m(w)) \le \textsc{lz}(h^{n(m)+1}(a))$$

   and $\textsc{lz}(h^n(a)) = \Theta(n)$, it follows that

$$\textsc{lz}_w(m) = \Theta(m^{1/e_a}).$$

2. $\rho_a > 1$ (h_a is exponential). There exist ρ_1 and ρ_2 positive numbers such that $\rho_1^n \le |h^n(a)| \le \rho_2^n$ which means that $n(m) = \Theta(\log m)$. By the same argument,

$$\textsc{lz}_w(m) = \Theta(\log m).$$

Notice however that h_a growing does not imply $\textsc{lz}_w(\cdot)$ unbounded. For instance, if $h(\mathsf{a}) = \mathsf{ab}$, $h(\mathsf{b}) = \mathsf{b}$, then h_{a} is polynomial but

$$w = h^{\infty}(\mathsf{a}) = \mathsf{abbb}\ldots$$

has bounded $\text{LZ}_w(\cdot)$. For the exponential case we can take $h(\mathsf{a}) = \mathsf{aa}$ whose fixed point has bounded Lempel–Ziv complexity as well.

Also, in the fist case above, we cannot have $e_a = 1$ as this implies bounded Lempel–Ziv complexity, contradicting the assumption on $\text{LZ}(h^n(a))$. Indeed, $e_a = 1$ implies $|h^n(a)| = \Theta(n)$ and so $|h^{n+1}(a)| - |h^n(a)|$ is bounded. Assuming $h(a) = au$, $u \neq \varepsilon$, we have $h^n(a) = auh(u)h^2(u) \cdots h^{n-1}(u)$. Consequently $|h^n(u)|$ is bounded hence we can find $h^n(u) = h^{n+p}(u)$ which implies $w = h^\infty(a)$ is ultimately periodic.

We have just proved the main result of the paper:

Theorem 2. *For a infinite fixed point $w = h^\omega(a)$ of a non-erasing morphism h, we have:*

1. *The Lempel–Ziv complexity of w is $\Theta(1)$ if and only if w is ultimately periodic.*
2. *If w is not ultimately periodic, then the Lempel–Ziv complexity of w is either $\Theta(\log n)$ or $\Theta(n^{1/k})$, $k \in \mathbb{N}$, $k \geq 2$, depending on whether h_a is exponential or polynomial, resp.*

We give next examples showing that all the above complexities are indeed possible.

Example 4. The highest Lempel–Ziv complexity is realized for $k = 2$, that means, $O(\sqrt{n})$, for the three letter morphism h_3 given by

$$h_3(\mathsf{a}) = \mathsf{ab}, \quad h_3(\mathsf{b}) = \mathsf{bc}, \quad h_3(\mathsf{c}) = \mathsf{c},$$

for which

$$h_3^n(\mathsf{a}) = \mathsf{abc}^0\mathsf{bc}^1 \ldots \mathsf{bc}^{n-1}.$$

Clearly, the growth function of a, $(h_3)_\mathsf{a}$, is quadratic whereas the complexity of powers is exactly linear which gives a final Lempel–Ziv complexity of $\sqrt{n}$; this can be checked directly by constructing the exhaustive history of $h_3^\infty(\mathsf{a})$:

$$(\mathsf{a}, \mathsf{b}, \mathsf{bc}, \mathsf{bc}^2, \ldots).$$

This example is extended to k letters by considering the morphism:

$$\begin{aligned}
&h_k : \{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_k\}^* \to \{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_k\}^*,\\
&h_k(\mathsf{a}_i) = \mathsf{a}_i\mathsf{a}_{i+1}, \text{ for all } 1 \leq i \leq k-1,\\
&h_k(\mathsf{a}_k) = \mathsf{a}_k.
\end{aligned}$$

We can use the theory of D0L growth functions to obtain that $(h_k)_{\mathsf{a}_1}$ is a polynomial of degree $k-1$ (see [26, Theorem 3.5]), or see that directly, as follows. Note that h_k restricted to $\{\mathsf{a}_2, \mathsf{a}_3, \ldots, \mathsf{a}_k\}^*$ is actually h_{k-1} modulo the renaming $\mathsf{a}_2 = \mathsf{a}_1, \mathsf{a}_3 = \mathsf{a}_2, \ldots, \mathsf{a}_k = \mathsf{a}_{k-1}$. Since

$$|(h_k)_{\mathsf{a}_1}(n)| = |\mathsf{a}_1\mathsf{a}_2 h(\mathsf{a}_2) \ldots h_k^{n-1}(\mathsf{a}_2)| = 1 + \sum_{i=0}^{n-1} |(h_{k-1})_{\mathsf{a}_1}(n)|\ ,$$

we conclude, by induction on $k \geq 3$, that $(h_{k-1})_{\mathsf{a}_1}(n) = \Theta(n^{k-2})$ implies $(h_k)_{\mathsf{a}_1}(n) = \Theta(n^{k-1})$. The base case follows from the previous example for $k = 3$.

Consequently, the Lempel–Ziv complexity of the infinite fixed point $h_k^\infty(\mathsf{a}_1)$ is $\Theta(\sqrt[k-1]{n})$. These examples illustrate the polynomial case.

Example 5. With respect to the exponential case, any uniform morphism with images of length k has a growth function of exactly k^n. Since the complexity of powers is linear for non-periodic words, the Lempel–Ziv complexity of the fixed point will be $\Theta(\log n)$.

Such an example is the famous Thue–Morse morphism, see Example 3, which fits the requirements for $k = 2$. Both of its infinite fixed points $t^\infty(\mathsf{a})$ and $t^\infty(\mathsf{b})$ are non-periodic and the growth functions associated with both letters are exactly 2^n. Their Lempel–Ziv complexity is $\Theta(\log n)$.

Example 6. Another famous example is given by the Fibonacci morphism

$$f(\mathsf{a}) = \mathsf{ab}, \quad f(\mathsf{b}) = \mathsf{a},$$

for which we can precisely compute the value of $\textsc{lz}(f^n(\mathsf{a})) = n + 1$. The powers of the Fibonacci morphism grow exponentially, at the rate $\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}$, and therefore the Lempel–Ziv complexity of the infinite word is again $\Theta(\log n)$.

6 Comparison with Factor Complexity

We dedicate the final section to a comparison between the Lempel–Ziv complexity and the factor complexity for infinite words generated by morphisms. The factor complexity is a natural function defined as the number of factors of a certain length occurring in an infinite word. For a word $w \in \Sigma^\omega$, this is

$$f_w(n) = \mathrm{card}(\{u \in \Sigma^* \mid u \in F(w), |u| = n\}) \ .$$

The investigation of factor complexity for the fixed points of morphisms has been initiated by Ehrenfeucht, Lee, and Rozenberg in [7] (they actually considered the closely related D0L-systems) and continued by Ehrenfeucht and Rozenberg in a series of papers, see [8,9,10,11,12,25]. The classification was completed by Pansiot [23,24] who found also the missing complexity class $\Theta(n \log\log n)$.

The following definitions appear, with different names, in [5]. The morphism h is called[1]

- *non-growing* if there exists a bounded letter in Σ,
- *u-exponential* if $\rho_a = \rho_b > 1$, $e_a = e_b = 1$, for all $a, b \in \Sigma$,

[1] What we call *u-*, *p-*, and *e-exponential* are *quasi-uniform*, *polynomially diverging*, and *exponentially diverging*, resp., in [23,24,5]. We changed the terminology so that it does not conflict with the corresponding notions for h_a.

- *p-exponential* if $\rho_a = \rho_b > 1$, for all a, b and $e_a > 1$, for some a, and
- *e-exponential* if $\rho_a > 1$, for all a and $\rho_a > \rho_b$, for some a, b.

Here is Pansiot's characterisation:

Theorem 3. *Let $w = h^\omega(a)$ be an infinite non-periodic word of factor complexity $f_w(\cdot)$.*

1. *If h is growing, then $f_w(n)$ is $\Theta(n)$, $\Theta(n \log\log n)$ or $\Theta(n \log n)$, depending on whether h is u-, p- or e-exponential, resp.*
2. *If h is not-growing, then either*
   (a) *w has arbitrarily large factors over the set of bounded letters and then $f_w(n) = \Theta(n^2)$ or*
   (b) *w has finitely many factors over the set of bounded letters and then $f_w(n)$ can be any of $\Theta(n)$, $\Theta(n \log\log n)$ or $\Theta(n \log n)$.*

In order to establish a correspondence with our hierarchy, we note that, in the first case of Theorem 3, the function h_a is exponential, which implies a logarithmic Lempel–Ziv complexity. However, a logarithmic Lempel–Ziv complexity does not necessarily imply one of the n, $n \log\log n$ or $n \log n$ cases for the factor complexity as it is illustrated by the following example.

Example 7. Consider the morphism h given by

$$h(\mathsf{a}) = \mathsf{abc}, \quad h(\mathsf{b}) = \mathsf{bac}, \quad h(\mathsf{c}) = \mathsf{c}.$$

Since h_a grows exponentially, $\textsc{lz}(h^\infty(\mathsf{a}))$ is, by Theorem 2, logarithmic. However, there exist arbitrarily large factors of $h^\infty(\mathsf{a})$ of the form c^n (c is bounded) which implies a $\Theta(n^2)$ factor complexity.

On the other hand, a radical-type Lempel–Ziv complexity does imply a quadratic factor complexity, as shown in the following lemma.

Lemma 7. *Assume $h : \Sigma^* \to \Sigma^*$ is a non-erasing morphism prolongeable on $a \in \Sigma$. If h_a is polynomial, then there exist arbitrarily large factors over Σ_B in $h^\infty(a)$.*

Therefore, Theorems 2 and 3, Example 7 and Lemma 7 imply the following result which gives the precise correspondence between the two complexities.

Theorem 4. *The correspondence between Lempel–Ziv and factor complexities for fixed points of morphisms is shown in Table 1, where all intersections are indeed possible.*

We see that both measures of complexity recognise ultimately periodic words as having bounded complexity, the lowest class of complexity.

In the nontrivial case of non-periodic fixed points, the Lempel–Ziv complexity groups together all words $h^\infty(a)$ with h_a exponential, whereas the factor complexity distinguishes four different complexities. On the other hand, the factor complexity does not make any distinction among words with h_a polynomial, whereas Lempel–Ziv gives an infinite hierarchy.

Table 1. Lempel–Ziv vs. factor complexity

	Lempel–Ziv complexity	Factor complexity
$h^\infty(a)$ is ultimately periodic	$\Theta(1)$	$\Theta(1)$
$h^\infty(a)$ is not ultimately periodic and h_a is polynomial	$\Theta(n^{\frac{1}{2}})$	$\Theta(n^2)$
	$\Theta(n^{\frac{1}{3}})$	
	$\vdots$	
	$\Theta(n^{\frac{1}{k}})$	
	$\vdots$	
$h^\infty(a)$ is not ultimately periodic and h_a is exponential	$\Theta(\log n)$	$\Theta(n^2)$
		$\Theta(n \log n)$
		$\Theta(n \log \log n)$
		$\Theta(n)$

References

1. J. M. Amigo, J. Szczepanski, E. Wajnryb, and M. V. Sanchez-Vives, Estimating the entropy rate of spike trains via Lempel-Ziv complexity, *Neural Computation* **16**(4) (2004) 717 – 736.
2. N.G. de Bruijn, A combinatorial problem, *Nederl. Akad. Wetensch. Proc.* **49** (1946) 758 – 764.
3. G. Chaitin, On the length of programs for computing finite binary sequences, *J. Assoc. Comput. Mach.* **13** (1966) 547 – 569.
4. X. Chen, S. Kwong and M. Li, A compression algorithm for DNA sequences, *IEEE Engineering in Medicine and Biology Magazine* **20**(4) (2001) 61 – 66.
5. C. Choffrut and J. Karhumäki, Combinatorics on words, in: G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages, Vol. I*, Springer-Verlag, Berlin, Heidelberg, 1997, 329 – 438.
6. M. Crochemore, Linear searching for a square in a word, in: A. Apostolico, Z. Galil, eds., *NATO Advanced Research Workshop on Combinatorial Algorithms on Words*, 1984, Springer-Verlag, Berlin, New York, 1985, 66 – 72.
7. A. Ehrenfeucht, K.P. Lee and G. Rozenberg, Subword complexities of various classes of deterministic developmental languages without interaction, *Theoret. Comput. Sci.* **1** (1975) 59 – 75.
8. A. Ehrenfeucht and G. Rozenberg, On the subword complexities of square-free D0L-languages, *Theoret. Comput. Sci.* **16** (1981) 25 – 32.
9. A. Ehrenfeucht and G. Rozenberg, On the subword complexities of D0L-languages with a constant distribution, *Theoret. Comput. Sci.* **13** (1981) 108 – 113.
10. A. Ehrenfeucht and G. Rozenberg, On the subword complexities of homomorphic images of languages, *RAIRO Informatique Théorique* **16** (1982) 303 – 316.
11. A. Ehrenfeucht and G. Rozenberg, On the subword complexities of locally catenative D0L-languages, *Information Processing Letters* **16** (1982) 7 – 9.

12. A. Ehrenfeucht and G. Rozenberg, On the subword complexities of m-free D0L-languages, *Information Processing Letters* **17** (1983) 121 – 124..
13. M. Farach, M.O. Noordewier, S.A. Savari, L.A. Shepp, A.D. Wyner, J. Ziv, On the entropy of DNA: algorithms and measurements based on memory and rapid convergence, *Proc. of SODA'95*, 1995, 48 – 57.
14. V.D. Gusev, V.A. Kulichkov, O.M. Chupakhina, The Lempel-Ziv complexity and local structure analysis of genomes, *Biosystems* **30**(1-3) (1993) 183 – 200.
15. L. Ilie, S. Yu and K. Zhang, Word complexity and repetitions in words, *Internat. J. Found. Comput. Sci.* **15**(1) (2004) 41 – 55.
16. A.N. Kolmogorov, Three approaches to the quantitative definition of information, *Probl. Inform. Transmission* **1** (1965) 1 – 7.
17. R. Kolpakov and G. Kucherov, Finding maximal repetitions in a word in linear time, *Proc. of the 40th Annual Symposium on Foundations of Computer Science*, IEEE Computer Soc., Los Alamitos, CA, 1999, 596 – 604.
18. A. Lempel and J. Ziv, On the Complexity of Finite Sequences , *IEEE Trans. Inform. Theory* **92**(1) (1976) 75 – 81.
19. M. Lothaire, *Combinatorics on Words*, Addison-Wesley, Reading, MA, 1983, (reprinted with corrections, Cambridge Univ. Press, Cambridge, 1997).
20. M. Lothaire, *Algebraic Combinatorics on Words*, Cambridge Univ. Press, 2002.
21. M. Lothaire, *Applied Combinatorics on Words*, Cambridge Univ. Press, 2005.
22. M.G. Main, Detecting leftmost maximal periodicities, *Discrete Appl. Math.* **25**(1-2) (1989) 145 – 153.
23. J.-J. Pansiot, Bornes inférieures sur la complexité des facteurs des mots infinis engendrés par morphismes itérés, *Proc. of STACS'84*, Lecture Notes in Comput. Sci. **166**, Springer, Berlin, 1984, 230 – 240.
24. J.-J. Pansiot, Complexité des facteurs des mots infinis engendrés par morphismes itérés, *Proc. of ICALP'84*, Lecture Notes in Comput. Sci. **172**, Springer, Berlin, 1984, 380 – 389.
25. G. Rozenberg, On subwords of formal languages, *Proc. of Fundamentals of computation theory*, Lecture Notes in Comput. Sci. **117**, Springer, Berlin-New York, 1981, 328 – 333.
26. G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, 1980.
27. W. Rytter, Application of Lempel-Ziv factorization to the approximation of grammar-based compression, *Theoret. Comput. Sci.* **302**(1-3) (2003) 211 – 222.
28. A. Salomaa and M. Soittola, *Automata-theoretic aspects of formal power series*, Springer, New York, 1978.
29. J. Szczepanski, M. Amigo, E. Wajnryb, and M.V. Sanchez-Vives, Application of Lempel-Ziv complexity to the analysis of neural discharges, *Network: Computation in Neural Systems* **14**(2) (2003) 335 – 350.
30. J. Szczepanski, J. M. Amigo, E. Wajnryb, and M. V. Sanchez-Vives, Characterizing spike trains with Lempel-Ziv complexity, *Neurocomputing* **58-60** (2004) 79 – 84.
31. J. Ziv and A. Lempel, A universal algorithm for sequential data compression, *IEEE Trans. Inform. Theory* **23**(3) (1977) 337 – 343.
32. J. Ziv and A. Lempel, Compression of individual sequences via variable-rate coding, *IEEE Trans. Inform. Theory* **24**(5) (1978) 530 – 536.

Partially Commutative Inverse Monoids

Volker Diekert, Markus Lohrey, and Alexander Miller

Universität Stuttgart, FMI, Germany
{diekert, lohrey, miller}@informatik.uni-stuttgart.de

Abstract. Free partially commutative inverse monoids are investigated. Analogously to free partially commutative monoids (trace monoids), free partially commutative inverse monoid are the quotients of free inverse monoids modulo a partially defined commutation relation on the generators. An $O(n \log(n))$ algorithm on a RAM for the word problem is presented, and NP-completeness of the generalized word problem and the membership problem for rational sets is shown. Moreover, free partially commutative inverse monoids modulo a finite idempotent presentation are studied. For these monoids, the word problem is decidable if and only if the complement of the commutation relation is transitive.

1 Introduction

A labelled transition system is deterministic, if in every state there is for each label at most one outgoing edge with this label. Of particular interest are systems where we can perform an undo-operation. This means that the system is codeterministic: For each state and label there is at most one incoming edge with this label. In this setting each label defines a partially defined injective mapping from states to states. The resulting transformations form an *inverse monoid*; and it is well-known that each inverse monoid arises this way. Because of its close connection to automata theory inverse monoids received quite an attention in theoretical computer science and there is a well-established literature on this subject, see e.g. [11,15].

In this paper we are interested in the situation where the labels describe actions of a (deterministic and codeterministic) transition system and some of the actions can be performed independently. This leads to a partial commutation and therefore to partially commutative inverse monoids. *Free partially commutative inverse monoids* were first studied in the thesis of da Costa [17], where, among others, the word problem has been shown to be decidable. Da Costa did not prove any complexity bounds. Our first contribution is a new approach to define free partially commutative inverse monoids which is closer to a standard construction of Margolis and Meakin [10]. We use a natural closure operation for subsets of *free partially commutative groups* (also known as *graph groups* [5]). Using our construction we are able to show in Section 3 that the word problem of a free partially commutative inverse monoid is solvable in time $O(n \log(n))$ on a RAM. In Section 4, we study the generalized word problem for free partially commutative inverse monoids. The generalized word problem asks whether a given monoid element belongs to a given finitely generated submonoid. In fact, we consider the more general membership problem for rational subsets of a free partially commutative inverse monoid, and we show its NP-completeness. NP-hardness appears already for the special case of the generalized word problem for a 2-generator free inverse monoid. It is quite remarkable that the generalized word problem remains decidable in our setting, because it is known to be undecidable for direct products of free groups [13]. So there is an undecidable problem for a direct product of free groups where the same problem is decidable for a direct product of free inverse monoids.

In the second part of the paper we consider free partially commutative inverse monoids modulo a finite idempotent presentations, which is a finite set of identities between idempotent elements. We show that the resulting quotient monoids have decidable word problems if and only if the underlying dependence structure is transitive. In the transitive case, the uniform word problem (where the idempotent presentation is part of the input) turns out to be EXPTIME-complete, whereas for a fixed idempotent presentation the word problem is solvable both in linear time on a RAM and logarithmic space on a Turing machine. These results generalize corresponding results for free inverse monoids modulo an idempotent presentation from [8,10]. Our decidability result for the case of a transitive dependence structure is unexpected in light of a result of Meakin and Sapir [12], where it was shown that there exist E-unitary inverse monoids over a finitely generated Abelian group, where the word problem is undecidable. The proof of this result in [12] is quite involved and relies on a sophisticated encoding of computations of Minski machines. In fact, a slight variation of our undecidability proof for non-transitive dependence structures can be used to give a simpler proof for the result of Meakin and Sapir; it will appear in the full version of this paper.

2 Preliminaries

In the following let Σ be a finite alphabet and $\Sigma^{-1} = \{a^{-1} \mid a \in \Sigma\}$ be a disjoint copy of Σ. We let $\Gamma = \Sigma \cup \Sigma^{-1}$. Then Γ becomes a set with an involution $^{-1} : \Gamma \to \Gamma$ by setting $(a^{-1})^{-1} = a$ for all $a \in \Sigma$. We extend this involution to an involution $^{-1} : \Gamma^* \to \Gamma^*$ by setting $(a_1 \cdots a_n)^{-1} = a_n^{-1} \cdots a_1^{-1}$ for $a_i \in \Gamma$, $1 \le i \le n$, $n \ge 0$. The *free group* generated by Σ is denoted by $F(\Sigma)$; it can be defined as the quotient monoid $\Gamma^*/\{aa^{-1} = 1 \mid a \in \Gamma\}$.

Let $\mathcal{M}$ be a finitely generated monoid and let Γ be a finite generating set for $\mathcal{M}$, i.e., there exists a surjective homomorphism $h : \Gamma^* \to \mathcal{M}$. The *word problem* for $\mathcal{M}$ is the computational problem that asks for two given words $u, v \in \Gamma^*$, whether $h(u) = h(v)$. The *generalized word problem* for $\mathcal{M}$ asks whether for given words $u, v_1, \ldots, v_n$ the element $h(u)$ belongs to the submonoid $\{h(v_1), \ldots, h(v_n)\}^* \subseteq \mathcal{M}$ generated by $h(v_1), \ldots, h(v_n)$. It is easy to see that the decidability/complexity of the (generalized) word problem does not depend on the underlying set of generators. Now let $\mathcal{G}$ be a finitely generated *group* and let Γ be a finite generating set for $\mathcal{G}$. The *Cayley graph* of $\mathcal{G}$ w.r.t. Γ is the undirected graph $\mathcal{C}(\mathcal{G}, \Gamma) = (\mathcal{G}, \{\{u, v\} \mid u^{-1}v \in \Gamma\})$. Note that the undirected edge $\{u, v\}$ can be viewed as a pair of directed edges (u, v), labelled with $u^{-1}v \in \Gamma$, and (v, u), labelled with $v^{-1}u \in \Gamma$.

2.1 Free Partially Commutative Inverse Monoids

A monoid $\mathcal{M}$ is called *inverse* if for every $x \in \mathcal{M}$ there is a *unique* $x^{-1} \in \mathcal{M}$ such that

$$xx^{-1}x = x \quad \text{and} \quad x^{-1}xx^{-1} = x^{-1}. \tag{1}$$

It is well-known that uniqueness of the inverse x^{-1} follows, if we require additionally to (1) that for all $x, y \in \mathcal{M}$:

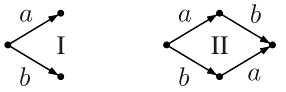
$$xx^{-1}yy^{-1} = yy^{-1}xx^{-1} \tag{2}$$

Elements xx^{-1} are exactly the idempotents in $\mathcal{M}$. It is clear that every mapping $\varphi : \Sigma \to \mathcal{M}$ to an inverse monoid $\mathcal{M}$ lifts uniquely to a homomorphism $\varphi : \Gamma^* \to \mathcal{M}$ such that $\varphi(u^{-1}) = \varphi(u)^{-1}$ for all $u \in \Gamma^*$. By an *independence relation* we mean here an irreflexive and symmetric relation $I_\Gamma \subseteq \Gamma \times \Gamma$ such that $(a, b) \in I_\Gamma$ implies $(a^{-1}, b) \in I_\Gamma$ for all $a, b \in \Gamma$. Note that I is specified by $I_\Sigma = I_\Gamma \cap \Sigma \times \Sigma$, and we may view (Σ, I_Σ) as an undirected graph. For words $u, v \in \Gamma^*$ we write $(u, v) \in I_\Gamma$ if $u = a_1 \cdots a_m, v = b_1 \cdots b_n$, and $(a_i, b_j) \in I_\Gamma$ for all $i \le m,\ j \le n$.

Every inverse monoid $\mathcal{M}$ can be viewed as a monoid of partially defined injections over a set Q. Thus, if $a \in \mathcal{M}$, then a is an injection $a : \mathrm{dom}(a) \hookrightarrow Q$ where $\mathrm{dom}(a) \subseteq Q$. Now partially defined injections a and b can be called *independent*, if the following three conditions are satisfied for all $q \in Q$:

(i) if $q \in \mathrm{dom}(a)$, then: $a(q) \in \mathrm{dom}(b) \iff q \in \mathrm{dom}(b)$,
(ii) if $q \in \mathrm{dom}(b)$, then: $b(q) \in \mathrm{dom}(a) \iff q \in \mathrm{dom}(a)$,
(iii) if $q \in \mathrm{dom}(a) \cap \mathrm{dom}(b)$, then: $ab(q) = ba(q)$.

These conditions look technical, but a brief reflection shows that they are indeed natural translations of an intuitive meaning of independence. Note that (i)–(iii) is a stronger requirement than to say that a and b commute. Consider the following situations:



In situation I the transitions a and b are not independent although they commute: The result of ab is the same as ba; it is the undefined mapping, which corresponds to a zero in the monoid. It is also clear that a and b should not be called independent in this situation because a can *disable* b (and vice versa). The situation II is different: The set Q has four states; it corresponds to the set of global states of the asynchronous product of two independent components. The first (second, resp.) component can only perform the action a (b, resp.). A simple calculation shows that if we define independence in $\mathcal{M}$ as above by (i)–(iii), then the independence of a and b implies the independence of a^{-1} and b, too. This motivates the following definition: An *inverse monoid over* (Σ, I_Σ) is an inverse monoid $\mathcal{M}$ together with a mapping $\varphi : \Sigma \to \mathcal{M}$ such that $\varphi(a)\varphi(b) = \varphi(b)\varphi(a)$ and $\varphi(a)^{-1}\varphi(b) = \varphi(b)\varphi(a)^{-1}$ for all $(a, b) \in I_\Sigma$. Thus, we can define the *free inverse monoid over* (Σ, I_Σ) by

$$\mathrm{FIM}(\Sigma, I) = \mathrm{FIM}(\Sigma)/\{ab = ba, a^{-1}b = ba^{-1} \mid (a, b) \in I_\Sigma\}.$$

Here, $\mathrm{FIM}(\Sigma)$ denotes the free inverse monoid over Σ, which is defined as the quotient monoid of $\Gamma^* = (\Sigma \cup \Sigma^{-1})^*$ modulo the equations in (1) and (2) for all $x, y \in \Gamma^*$. In the following, we will briefly write I for both I_Σ and I_Γ. It will be always clear, on which set I is defined. The monoid $\mathrm{FIM}(\Sigma, I)$ is also called a *free partially commutative inverse monoid*.

Da Costa has studied $\mathrm{FIM}(\Sigma, I)$ in his thesis from a more general viewpoint of graph products [17]. As a consequence he showed that $\mathrm{FIM}(\Sigma, I)$ has a decidable word problem. In his construction he used the general approach via Schützenberger graphs and Stephen's iterative procedure [16]. The decidability of the word problem follows because da Costa can show that Stephen's procedure terminates. However, no complexity bounds are given in [17].

Another starting point for defining free partially commutative inverse monoids is a construction of Margolis and Meakin [9]. One would start with the free partially commutative group $G(\Sigma, I)$ (defined below) and consider as elements of an inverse monoid the pairs (A, g), where A is a finite and connected subgraph of the Cayley graph of $G(\Sigma, I)$ with $1, g \in A$. Although this construction yields for $I = \emptyset$ indeed $\mathrm{FIM}(\Sigma)$ by a result of Munn [14], it fails for $I \neq \emptyset$, simply because independent generators do not commute. Thus, we have to do something else. Fortunately it is enough to modify the construction of Margolis and Meakin slightly in order to achieve a simple and convenient description of the elements in $\mathrm{FIM}(\Sigma, I)$. Our approach is based on the notion of coherently prefix-closed subsets which we make precise in the next section.

2.2 Trace Monoids and Graph Groups

Recall that $\Gamma = \Sigma \cup \Sigma^{-1}$ and $I \subseteq \Gamma \times \Gamma$ is an irreflexive and symmetric relation such that $(a, b) \in I$ implies $(a^{-1}, b) \in I$. Let $M(\Gamma, I) = \Gamma^* / \{ab = ba \mid (a, b) \in I\}$ be the *free partially commutative monoid* (or *trace monoid*) over (Γ, I). Due to $(a, b) \in I \Rightarrow (a^{-1}, b) \in I$, the involution $^{-1} : \Gamma^* \to \Gamma^*$ is well-defined on $M(\Gamma, I)$. The relation $D = (\Gamma \times \Gamma) \setminus I$ is called the *dependence relation*.

There is a rich theory on trace monoids [4]. Here we need some basic results, only. The perhaps most important fact ist that traces (i.e., elements of $M(\Gamma, I)$) have a unique description as *dependence graphs*, which are node-labelled acyclic graphs. Let $u = a_1 \cdots a_n \in \Gamma^*$ be a word. The vertex set of the dependence graph of u is $\{1, \ldots, n\}$ and vertex i is labelled with $a_i \in \Gamma$. There is an edge from vertex i to j if and only if $i < j$ and $(a_i, a_j) \in D$. Now, two words define the same trace in $M(\Gamma, I)$ if and only if their dependence graphs are isomorphic.

A *clique covering* of the dependence relation D is a tuple $(\Gamma_i)_{1 \leq i \leq k}$ such that $\Gamma = \bigcup_{i=1}^{k} \Gamma_i$ and $D = \bigcup_{i=1}^{k} \Gamma_i \times \Gamma_i$. W.l.o.g. we may assume that $a \in \Gamma_i$ if and only if $a^{-1} \in \Gamma_i$. Let $\pi_i : M(\Gamma, I) \to \Gamma_i^*$ the *projection homomorphism* which deletes all letters from $\Gamma \setminus \Gamma_i$. The morphism $\pi : M(\Gamma, I) \to \prod_{i=1}^{k} \Gamma_i^*$ defined by $\pi(u) = (\pi_1(u), \ldots, \pi_k(u))$ is injective [2,3]. For $u, v \in M(\Gamma, I)$ we write $u \leq v$ if u is a prefix of v, i.e., $v = uw$ in $M(\Gamma, I)$ for some trace w. A trace f is a *factor* of u, if we can write $u = pfq$ in $M(\Gamma, I)$. Let $\max(u) = \{a \in \Gamma \mid u = ta \text{ for some trace } t\}$; it is the set of labels of the maximal nodes in the dependence graph of u.

An important fact about traces is the following: Assume we have $u \leq w$ and $v \leq w$ for some $u, v, w \in M(\Gamma, I)$. Then the supremum $u \sqcup v \in M(\Gamma, I)$ w.r.t. the prefix order $\leq$ exists. We can define $u \sqcup v$ by restricting the dependence graph of w to the domain of u and v, where u and v are viewed as downward-closed subsets of the dependence graph of w. If $(\Gamma_i)_{1 \leq i \leq k}$ is a clique covering of the dependence relation, then for every i, either $\pi_i(u) \leq \pi_i(v)$ and $\pi_i(u \sqcup v) = \pi_i(v)$ or $\pi_i(v) \leq \pi_i(u)$ and $\pi_i(u \sqcup v) = \pi_i(u)$. A trace p is a *prime* if $|\max(p)| = 1$.

For $t \in M(\Gamma, I)$ let $\mathbb{P}(t) = \{p \leq t \mid p \text{ is a prime}\}$. Note that $t = \sqcup \mathbb{P}(t)$ (the supremum of the traces in $\mathbb{P}(t)$). Let $A \subseteq M(\Gamma, I)$. We define $\mathbb{P}(A) = \bigcup_{t \in A} \mathbb{P}(t)$. The set A is called *prefix-closed*, if $u \leq v \in A$ implies $u \in A$. It is called *coherently-closed* if for every $C \subseteq A$ such that $\sqcup C$ exists, $\sqcup C \in A$. One can show that A is coherently-closed if for all $u, v \in A$ such that $u \sqcup v$ exists, $u \sqcup v \in A$. In the following we say that A is *closed*, if it is both prefix-closed and coherently-closed. Clearly, for every $A \subseteq M(\Gamma, I)$ there is a smallest closed set $\overline{A}$ with $A \subseteq \overline{A}$. One has $\overline{A} = \{\sqcup C \mid C \text{ is a set of prefixes of } A, \sqcup C \text{ exists}\}$ and $\overline{A} = \overline{\overline{A}}$. The notions of a prime and coherence are standard in domain theory and the connection to trace theory is exposed in [4, Sec. 11.3].

Lemma 1. *For $A, B \subseteq M(\Gamma, I)$ we have $\overline{A} = \overline{B}$ if and only if $\mathbb{P}(A) = \mathbb{P}(B)$.*

A *trace rewriting system* R over $M(\Gamma, I)$ is just a finite subset of $M(\Gamma, I) \times M(\Gamma, I)$ [3]. We can define the one-step rewrite relation $\rightarrow_R \subseteq M(\Gamma, I) \times M(\Gamma, I)$ by: $x \rightarrow_R y$ if and only if there are $u, v \in M(\Gamma, I)$ and $(\ell, r) \in R$ such that $x = u\ell v$ and $y = urv$. The notion of a *confluent* and *terminating* trace rewriting system is defined as for other types of rewriting systems. A trace u is an *irreducible normal form* of t if $t \xrightarrow{*}_R u$ and there does not exist a trace v with $u \rightarrow_R v$.

The *free partially commutative group* (or *graph group* [5]) over (Σ, I), briefly $G(\Sigma, I)$, is the quotient of the free group $F(\Sigma)$ modulo the defining relations $ab = ba$ for all $(a, b) \in I$. Clearly, $G(\Sigma, I) = M(\Gamma, I)/\{aa^{-1} = 1 \mid a \in \Gamma\}$. We can define a confluent and terminating trace rewriting system $R = \{aa^{-1} \rightarrow 1 \mid a \in \Gamma\}$, where $1 \in M(\Gamma, I)$ denotes the empty trace. Given $u \in \Gamma^*$ we can view $u \in M(\Gamma, I)$ and compute its irreducible normal form $\widehat{u} \in M(\Gamma, I)$ w.r.t. R in linear time [3]. Thus, $\widehat{u}$ is a trace without any factor of the form aa^{-1} for $a \in \Gamma$. We also say that the trace $\widehat{u}$ is *reduced*. We have $u = v$ in $G(\Sigma, I)$ if and only if $\widehat{u} = \widehat{v}$. This allows to solve the word problem in $G(\Sigma, I)$ in linear time [3,19]. In the following, whenever $u \in \Gamma^*$ (or $u \in M(\Gamma, I)$ or $u \in G(\Sigma, I)$), then $\widehat{u} \in M(\Gamma, I)$ denotes this unique reduced trace such that $u = \widehat{u}$ in $G(\Sigma, I)$. The set $\widehat{M}(\Gamma, I) = \{\widehat{u} \mid u \in M(\Gamma, I)\}$ is in canonical one-to-one correspondence with $G(\Sigma, I)$, hence we may identify $\widehat{u}$ with the group element it represents.

A subset $A \subseteq G(\Sigma, I)$ is called *closed*, if the set of reduced traces $\widehat{A} = \{\widehat{g} \in \widehat{M}(\Gamma, I) \mid g \in A\}$ is closed. Clearly, for every $A \subseteq G(\Sigma, I)$, there is a smallest closed subset $\overline{A} \subseteq G(\Sigma, I)$ such that $A \subseteq \overline{A}$. We have $\overline{A} = \overline{\overline{A}}$ and we can identify $\overline{A}$ with $\overline{\widehat{A}} \subseteq \widehat{M}(\Gamma, I)$. Note that $\widehat{M}(\Gamma, I)$ is closed. Recall that I is irreflexive, hence $\{1, a, a^{-1}\}$ is closed since $(a, a^{-1}) \in D$. We now give a geometric interpretation of closed sets. Let $g, h \in G(\Sigma, I)$. A *geodesic* between g and h is a shortest path in the Cayley graph of $G(\Sigma, I)$. The labelling of such a path is unique as a reduced trace $\widehat{u} \in \widehat{M}(\Gamma, I)$ such that $g\widehat{u} = h$ in $G(\Sigma, I)$. We say that $f \in G(\Sigma, I)$ is on a geodesic from g to h if $\widehat{f} \leq \widehat{u}$.

Proposition 1. *A subset $A \subseteq G(\Sigma, I)$ is closed if and only if both $1 \in A$ and whenever f is on a geodesic from g to h with $g, h \in A$, then $gf \in A$, too.*

Corollary 1. *Let $A \subseteq G(\Sigma, I)$ be closed and $g \in A$. Then $g^{-1}A$ is closed, too.*

Proof. Since $g \in A$, we have $1 \in g^{-1}A$. The property "f is on a geodesic from h_1 to h_2 with $h_1, h_2 \in A$ implies $h_1 f \in A$" is invariant by translation. Thus A satisfies this property if and only if $g^{-1}A$ satisfies this property. □

2.3 A Realization of Free Partially Commutative Inverse Monoids

We are now ready to give a concrete realization of the free inverse monoid over (Σ, I). The realization is very much in the spirit of Margolis and Meakin [9], but differs in the subtle point that we allow closed subsets of $G(\Sigma, I)$, only. Consider the set of pairs (A, g) where $A \subseteq G(\Sigma, I)$ is a finite and closed subset of the graph group $G(\Sigma, I)$ and $g \in A$. This set becomes a monoid by

$$(A, g) \cdot (B, h) = (\overline{A \cup gB}, gh).$$

An immediate calculation shows that the operation is associative and that $(\{1\}, 1)$ is a neutral element. Moreover, the idempotents are the elements of the form $(A, 1)$ and idempotents commute. By Corollary 1, if $g \in A \subseteq G(\Sigma, I)$ and A is closed, then $g^{-1}A$ is closed, too. Hence we can define $(A, g)^{-1} = (g^{-1}A, g^{-1})$. A simple calculation shows that (1) and (2) are satisfied. Thus our monoid is an inverse monoid. We view Γ as a subset of this monoid by identifying $a \in \Gamma$ with the pair $(\overline{\{1, a\}}, a)$, and this yields a canonical homomorphism γ defined by $\gamma(u) = (\overline{\{\widehat{v} \mid v \leq u\}}, \widehat{u})$ for $u \in \Gamma^*$. We obtain $\gamma(ab) = (\{1, a\}, a) \cdot (\{1, b\}, b) = (\overline{\{1, a, ab\}}, ab)$. Now, if $(a, b) \in I$, then $\overline{\{1, a, ab\}} = \{1, a, b, ab\} = \overline{\{1, b, ba\}}$, i.e., $\gamma(ab) = \gamma(ba)$. Hence, we obtain an inverse monoid over (Σ, I) since $(a, b) \in I$ implies $(a^{-1}, b) \in I$. As a consequence, the homomorphism γ can be viewed as a canonical homomorphism

$$\gamma : \mathrm{FIM}(\Sigma, I) \to \{\, (A, g) \mid g \in A \subseteq G(\Sigma, I),\ A \text{ finite and closed}\,\}. \tag{3}$$

Theorem 1. *The morphism γ in (3) is an isomorphism.*

Proof. Consider a pair (A, g) with $A \subseteq G(\Sigma, I)$ finite and closed and $g \in A$. Recall that $\widehat{A} = \{\widehat{u} \in \widehat{M}(\Gamma, I) \mid u \in A\}$. Let $w \in \Gamma^*$ be an arbitrary word representing the trace $(\prod_{\widehat{u} \in \widehat{A}} \widehat{u}\,\widehat{u}^{-1})\widehat{g}$ where the product is taken in any order. Then a simple reflection shows $\gamma(w) = (A, g)$. Hence γ is surjective. It remains to show that γ is injective. To see this let $w \in \Gamma^*$ and $\gamma(w) = (A, g)$. Note that $\widehat{w} = \widehat{g}$. It suffices to show

$$w = (\prod_{u \in \widehat{A}} uu^{-1})\widehat{w} \ \text{ in } \mathrm{FIM}(\Sigma, I). \tag{4}$$

This is enough because it implies $\gamma(w) = \gamma(w') \implies w = w'$ in $\mathrm{FIM}(\Sigma, I)$ for all $w, w' \in \Gamma^*$. If $w = \varepsilon$ then $(A, g) = (\{1\}, 1)$ and (4) is true. Hence let $w = va$ with $a \in \Gamma$. By induction $v = (\prod_{u \in \widehat{B}} uu^{-1})\widehat{v}$ in $\mathrm{FIM}(\Sigma, I)$, where $\gamma(v) = (B, h)$. We obtain $\widehat{A} = \overline{\widehat{B} \cup \{\widehat{w}\}}$ and $w = (\prod_{u \in \widehat{B}} uu^{-1})\widehat{v}a$ in $\mathrm{FIM}(\Sigma, I)$.

We distinguish whether $\widehat{v}a$ is reduced or not. If $\widehat{v}a$ is not reduced, then $\widehat{v} = \widehat{w}a^{-1} \in \widehat{B}$, i.e., $\widehat{w} \in \widehat{B}$ since $\widehat{B}$ is prefix-closed. It follows $\widehat{B} = \widehat{A}$. We obtain in $\mathrm{FIM}(\Sigma, I)$:

$$w = (\prod_{u \in \widehat{B}} uu^{-1})\widehat{v}a = (\prod_{u \in \widehat{A}} uu^{-1})\widehat{w}a^{-1}a = (\prod_{u \in \widehat{A}} uu^{-1})\widehat{w}\,\widehat{w}^{-1}\widehat{w}\,a^{-1}a$$

$$= (\prod_{u \in \widehat{A}} uu^{-1})\widehat{w}a^{-1}a\widehat{w}^{-1}\widehat{w} = (\prod_{u \in \widehat{A}} uu^{-1})\widehat{v}\widehat{v}^{-1}\widehat{w} = (\prod_{u \in \widehat{A}} uu^{-1})\widehat{w}.$$

It remains the case where $\widehat{va} = \widehat{v}a = \widehat{w}$. We obtain in $\mathrm{FIM}(\Sigma, I)$:

$$w = (\prod_{u \in \widehat{B}} uu^{-1})\widehat{w} = (\prod_{u \in \widehat{B}} uu^{-1})\widehat{w}\widehat{w}^{-1}\widehat{w}.$$

Clearly, $\widehat{w} \in \widehat{A}$. Hence $w = (\prod_{u \in A'} uu^{-1})\widehat{w}$ in $\mathrm{FIM}(\Sigma, I)$ for some subset $A' \subseteq \widehat{A}$ such that $\overline{A'} = \widehat{A}$ (set $A' = \widehat{B} \cup \{\widehat{w}\}$). Therefore it is enough to show $\prod_{u \in A'} uu^{-1} = \prod_{u \in \widehat{A}} uu^{-1}$ in $\mathrm{FIM}(\Sigma, I)$. This is the assertion of the following claim.

Claim: Let $A \subseteq M(\Gamma, I)$. Then $\prod_{u \in A} uu^{-1} = \prod_{u \in \overline{A}} uu^{-1}$ in $\mathrm{FIM}(\Sigma, I)$.

To prove this claim, let $v \leq u \in A$. Then $u = vw = vv^{-1}vw = vv^{-1}u$ in $\mathrm{FIM}(\Sigma, I)$. Hence we may assume that A is prefix closed. Now, let $u, v \in A$ such that $w = u \sqcup v$ exists. Then, by Levi's lemma (see e.g. [4, p. 10]), $u = pr$, $v = ps$, and $w = psr$ with $(r, s) \in I$. We obtain in $\mathrm{FIM}(\Sigma, I)$: $uu^{-1}vv^{-1} = prr^{-1}p^{-1}pss^{-1}p^{-1} = prr^{-1}ss^{-1}p^{-1} = prss^{-1}r^{-1}p^{-1} = ww^{-1}$. This means that we may assume that A is coherently-closed, too. But if A is both prefix-closed and coherently-closed, then $A = \overline{A}$ by definition of $\overline{A}$. Hence the claim and the theorem follow. □

For $I = \emptyset$, Theorem 1 yields Munn's theorem [14] as a special case. Note that for $I = \emptyset$, the closure $\overline{A}$ of a prefix-closed subset A of the free group $F(\Sigma)$ equals A itself. It follows that $\gamma(u) = (\{\widehat{v} \mid v \leq u\}, \widehat{u})$, where $\widehat{v} \in \Gamma^*$ is the unique irreducible word corresponding to $v \in \Gamma^*$. The set $\{\widehat{v} \mid u \leq v\}$ is also called the *Munn tree* of u.

Since we are interested in computational problems, we are concerned with the input size of elements in $\mathrm{FIM}(\Sigma, I)$. The standard representation is just a word u over the alphabet Γ. If $\gamma(u) = (A, g)$, then $|A| \leq |u|^k$, where k is the number of cliques in a clique covering for the dependence relation, because $t \in \widehat{A}$ implies that $\pi_i(t)$ is a prefix of $\pi_i(u)$ for all $1 \leq i \leq k$. Hence, for a fixed (Σ, I), the size of A is bounded polynomially in the length of u, and moreover A can be calculated in polynomial time from u. Thus, for computational problems in or above polynomial time, we can represent $(A, g) \in \mathrm{FIM}(\Sigma, I)$ by listing all the elements of A followed by g. In fact, instead of writing down the closed set A, it suffices to list the primes in $\mathbb{P}(A)$ by Lemma 1. The set $\mathbb{P}(A)$ has size at most $|u|$ whatever (Σ, I) is. But in general, the more concise representation is still the standard representation.

3 The Word Problem in $\mathrm{FIM}(\Sigma, I)$

Using Munn's theorem [14], it is easy to solve the word problem for a free inverse monoid in linear time on a RAM. For free partially commutative inverse monoids the solution of the word problem is more involved. We are able to present an $O(n \log(n))$-algorithm on a RAM by using a sophisticated combination of simple data structures:

Theorem 2. *For a fixed free partially commutative inverse monoid* $\mathrm{FIM}(\Sigma, I)$*, the word problem can be solved in time* $O(n \log(n))$ *on a RAM.*

Proof. Let $u, v \in \Gamma^*$. By [3,19], we can test in linear time whether $u = v$ in $G(\Sigma, I)$. It remains to check equality of the closures. Let $(\Gamma_i)_{1 \le i \le k}$ be a clique covering of the dependence relation $D = (\Gamma \times \Gamma) \setminus I$. With $w \in \Gamma^*$ we associate the following data:

– the prefix-closed set of words $T_i(w) = \{\pi_i(\widehat{s}) \mid s \le w\}$,
– the word $p_i(w) = \pi_i(\widehat{w}) \in T_i$ for every $1 \le i \le k$,
– the set of primes $P(w) = \mathbb{P}(\{\widehat{s} \mid s \le w\})$
– the linearly ordered (w.r.t. the prefix order) set $P_i(w) = \mathbb{P}(\widehat{w}) \cap \{p \mid \max(p) \in \Gamma_i\}$.

By Lemma 1, we have to check whether $P(u) = P(v)$. Before we present an efficient implementation of the data structures above, let us first show how to compute $(T_i(wa), p_i(wa), P_i(wa))_{1 \le i \le k}$, $P(wa)$ from $(T_i(w), p_i(w), P_i(w))_{1 \le i \le k}$, $P(w)$ for $a \in \Gamma$. For this, we have to distinguish the two cases $a^{-1} \notin \max(\widehat{w})$ and $a^{-1} \in \max(\widehat{w})$. We use the following notation: For $a \in \Gamma$ such that a occurs in t define the prime $\delta_a(t)$ as the maximal prefix of t such that $\max(\delta_a(t)) = \{a\}$. In case that a does not occur in t let $\delta_a(t) = 1$. We obtain $\delta_a(ta) = (\sqcup\{\delta_b(t) \mid (a, b) \in D\})\, a$.

Case 1. $a^{-1} \notin \max(\widehat{w})$, i.e., $\widehat{wa} = \widehat{w}a$. We have:

$$T_i(wa) = \begin{cases} T_i(w) \cup \{p_i(w)a\} & \text{if } a \in \Gamma_i \\ T_i(w) & \text{otherwise} \end{cases} \qquad p_i(wa) = \begin{cases} p_i(w)a & \text{if } a \in \Gamma_i \\ p_i(w) & \text{otherwise} \end{cases}$$

$$P_i(wa) = \begin{cases} P_i(w) \cup \{\delta_a(\widehat{w}a)\} & \text{if } a \in \Gamma_i \\ P_i(w) & \text{otherwise} \end{cases} \qquad P(wa) = P(w) \cup \{\delta_a(\widehat{w}a)\}$$

Note that

$$\delta_a(\widehat{w}a) = (\sqcup\{\delta_b(\widehat{w}) \mid (a, b) \in D\})\, a = (\sqcup\{\max P_i(w) \mid 1 \le i \le k, a \in \Gamma_i\})\, a.$$

Case 2. $a^{-1} \in \max(\widehat{w})$, i.e., $\widehat{w} = sa^{-1}$ and $\widehat{wa} = s$ for an irreducible trace $s \in \widehat{M}(\Gamma, I)$. We have $\max P_i(w) = \delta_{a^{-1}}(\widehat{w})$ for all i with $a^{-1} \in \Gamma_i$ (i.e., $a \in \Gamma_i$) and:

$$T_i(wa) = T_i(w) \qquad p_i(wa) = \begin{cases} v & \text{if } a^{-1} \in \Gamma_i,\, p_i(w) = va^{-1} \\ p_i(w) & \text{otherwise} \end{cases}$$

$$P(wa) = P(w) \cup \mathbb{P}(s) = P(w) \qquad P_i(wa) = \begin{cases} P_i(w) \setminus \max P_i(w) & \text{if } a^{-1} \in \Gamma_i \\ P_i(w) & \text{otherwise} \end{cases}$$

For the equality $P(w) \cup \mathbb{P}(s) = P(w)$ note that $\mathbb{P}(s) \subseteq P(w)$ since the trace s is a prefix of the trace $\widehat{w}$. Let us now discuss an efficient implementation of our data structures such that the updates above can be done in time $O(\log(n))$. The prefix-closed set $T_i(w)$ can be stored as a trie with at most $|\pi_i(w)|$ many nodes, i.e., a rooted tree, where every node has for every $a \in \Gamma_i$ at most one a-labelled outgoing edge and $T_i(w)$ equals the set of all path-labels from the root to tree nodes. We assign with every node of $T_i(w)$ a key from $\mathbb{N}$. The root gets the key 1, and with every new node of $T_i(w)$ the key is increased by one. This allows to calculate $\max U$ for a subset $U \subseteq T_i(w)$, which is linearly ordered by the prefix relation, in time $O(|U|)$ by comparing the keys for the nodes in U. The word $p_i(w) = \pi_i(\widehat{w})$ is just a distinguished node of the trie

$T_i(w)$. Clearly, $a^{-1} \in \max(\widehat{w})$ if and only if $p_i(w)$ ends with a^{-1} for all $1 \leq i \leq k$ with $a^{-1} \in \Gamma_i$. This means that whenever $a^{-1} \in \Gamma_i$, then $p_i(w)$ is the a^{-1}-successor of its parent node. This allows to distinguish between case 1 and case 2 above in time $O(k) = O(1)$. In case 1, we have to add an a-successor to the node $p_i(w)$ in case $a \in \Gamma_i$ and $p_i(w)$ does not have an a-successor yet. This new node becomes $p_i(wa)$. If $a \in \Gamma_i$ but $p_i(w)$ already has an a-successor v, then v becomes $p_i(wa)$. In case 2, the tries do not change, but if $a \in \Gamma_i$, then $p_i(wa)$ is the father node of $p_i(w)$.

The set of primes $P(w)$ is stored as the set of tuples $\{(\pi_i(t))_{1 \leq i \leq k} \mid t \in P(w)\}$, where every projection $\pi_i(t)$ is represented by the corresponding node in the trie $T_i(w)$. For the set $P(w)$ we use a data structure, which allows $O(\log(n))$ time implementations for the operations insert and find. The linearly ordered set $P_i(w)$ is stored as a list of tuples $(\pi_i(t))_{1 \leq i \leq k}$ for $t \in P_i(w)$. Using this representation, the necessary updates for case 2 are clearly possible in constant time. For case 1, we have to calculate the tuple corresponding to $\delta_a(\widehat{wa}) = (\sqcup\{\max P_i(w) \mid 1 \leq i \leq k, a \in \Gamma_i\})\, a$. Note that

$$\pi_j(\delta_a(\widehat{wa})) = \sqcup\{\pi_j(\max P_i(w)) \mid 1 \leq i \leq k, a \in \Gamma_i\}\, \pi_j(a) \tag{5}$$

for $1 \leq j \leq k$. The $\sqcup$ in (5) refers to the prefix order on words. Note that since $\max P_i(w)$ is a prefix of the trace $\widehat{w}$ for every i, the set $\{\pi_j(\max P_i(w)) \mid 1 \leq i \leq k, a \in \Gamma_i\}$ is linearly ordered by the prefix relation on Γ_j^*, i.e., the supremum exists. Moreover, this supremum can be computed in time $O(k)$, where k is the number of cliques (which is a constant) by using the keys associated with the nodes from $T_j(w)$. This concludes the description of our data structures. Now for our input words $u, v \in \Gamma^*$ we first compute $(T_i(u), p_i(u), P_i(u))_{1 \leq i \leq k}$, $P(u)$ and $(T_i(v), p_i(v), P_i(v))_{1 \leq i \leq k}$, $P(v)$ in time $O(n \log(n))$. When building up the tries $T_i(u)$ and $T_i(v)$ we have to use the same node name for a certain string over Γ_i. Then we can check $P(u) = P(v)$ in time $O(n \log(n))$ using the set data structures for $P(u)$ and $P(v)$. $\square$

For the uniform word problem, where the independence relation I is part of the input, the above algorithm still yields a polynomial time algorithm. More precisely, the running time is $O((k^2 + \log(n))n)$, where k is the number of cliques in a clique covering for the dependence relation and n is the length of the input words.

4 The Generalized Word Problem in FIM(Σ, I)

In this section, we show that the generalized word problem for a free partially commutative inverse monoid is NP-complete in general. An NP upper bound can be even shown for the membership problem in rational sets of $\mathrm{FIM}(\Sigma, I)$. A rational subset of $\mathrm{FIM}(\Sigma, I)$ is represented concisely by a finite automaton over the alphabet Γ.

Theorem 3. *For every fixed free partially commutative inverse monoid* $\mathrm{FIM}(\Sigma, I)$*, the membership problem for rational subsets of* $\mathrm{FIM}(\Sigma, I)$ *belongs to* NP.

Proof. For given $(B, g) \in \mathrm{FIM}(\Sigma, I)$ and a finite automaton $\mathcal{A}$ over the alphabet Γ we have to determine whether $(B, g) \in \gamma(L(\mathcal{A}))$, where $\gamma : \Gamma^* \to \mathrm{FIM}(\Sigma, I)$ is the canonical morphism. In the following, we view B as a subgraph of the Cayley graph of $G(\Sigma, I)$. In a first step we guess a connected subset $C \subseteq B$ with $1, g \in C$ such that its closure $\overline{C}$ equals B. It remains to check in NP whether there is a path from 1 to g in C, which visits all nodes of C and such that this path is labelled with a word from $L(\mathcal{A})$.

Let n be the number of states of the automaton $\mathcal{A}$. Assume that $p = (v_1, \ldots, v_m)$ is a path in C such that $v_1 = 1$, $v_m = g$, $C = \{v_1, \ldots, v_m\}$ and let $q_1 \ldots, q_m$ be a corresponding path in the automaton $\mathcal{A}$, where q_1 is the initial state and q_m is a final state. Let $i_1 < \cdots < i_\ell$ be exactly those positions $j \in \{2, \ldots, m\}$ such that $v_j \notin \{v_1, \ldots, v_{j-1}\}$. Clearly, $\ell < |C|$. Set $i_0 = 1$ and $i_{\ell+1} = m + 1$. Assume that $|i_{k+1} - i_k| > |C| \cdot n$ for some $k \in \{0, \ldots, \ell\}$. Then there are positions $i_k \leq \alpha < \beta < i_{k+1}$ such that $v_\alpha = v_\beta$ and $q_\alpha = q_\beta$. It follows that $v_1, \ldots, v_\alpha, v_{\beta+1}, \ldots, v_m$ is a again a path in C from 1 to g, which visits all nodes of C, and $q_1, \ldots, q_\alpha, q_{\beta+1}, \ldots, q_m$ is a corresponding path in the automaton $\mathcal{A}$.

From the above consideration it follows that if there exists a path from 1 to g in C, which visits all nodes of C and such that this path is labelled with a word from $L(\mathcal{A})$, then there exists such a path of length at most $|C|^2 \cdot n$. Such a path can be guessed in polynomial time. This finishes the proof. □

NP-hardness can be already shown for the generalized word problem of $\mathrm{FIM}(\{a, b\})$:

Theorem 4. *The generalized word problem for* $\mathrm{FIM}(\{a, b\})$ *is* NP-*hard.*

Proof. We prove the theorem by a reduction from SAT. Let $\Psi = \{C_1, \ldots, C_m\}$ be a set of clauses over variables $x_1, \ldots, x_n$. Let $k = m + n$. For $1 \leq i \leq n$ let $P_i = \{j \mid x_i \in C_j\}$ and $N_i = \{j \mid \neg x_i \in C_j\}$. Let $u = (A, a^n) \in \mathrm{FIM}(\{a, b\})$, where (the subgraph of the Cayley graph of $F(\{a, b\})$ induced by) A looks as follows:

$$\begin{tikzcd}
1 \arrow[r, "a"] & a \arrow[r, "a"] \arrow[d, "b"] & a^2 \arrow[d, "b"] & \cdots & a^{m-1} \arrow[r, "a"] \arrow[d, "b"] & a^m \arrow[r, "a"] \arrow[d, "b"] & a^{m+1} & \cdots & a^{k-1} \arrow[r, "a"] & a^k \\
 & ab & a^2b & & a^{m-1}b & a^mb & & & &
\end{tikzcd}$$

The idea is that the node a^jb represents the clause C_j. For every $1 \leq i \leq n$ define $u_{i,t} = (A_{i,t}, a) \in \mathrm{FIM}(\{a, b\})$ and $u_{i,f} = (A_{i,f}, a) \in \mathrm{FIM}(\{a, b\})$, where:

$$A_{i,t} = a^{-i+1}(\{1, a, \ldots, a^k\} \cup \{a^jb \mid j \in P_i\}) \subseteq F(\{a, b\})$$
$$A_{i,f} = a^{-i+1}(\{1, a, \ldots, a^k\} \cup \{a^jb \mid j \in N_i\}) \subseteq F(\{a, b\})$$

We claim that $u \in \{u_{1,t}, u_{1,f}, \ldots, u_{n,t}, u_{n,f}\}^*$ if and only if Ψ is satisfiable. First assume that Ψ is satisfied and let $\sigma : \{x_1, \ldots, x_n\} \to \{\text{true}, \text{false}\}$ be a satisfying assignment. Let $u_i = u_{i,t}$ if $\sigma(x_i) = $ true, otherwise set $u_i = u_{i,f}$. Then we have $u = u_1 \cdots u_n$, which shows $u \in \{u_{1,t}, u_{1,f}, \ldots, u_{n,t}, u_{n,f}\}^*$. For the other direction assume that $u = u_1 \cdots u_m$, where $m \geq 0$ and $u_1, \ldots, u_m \in \{u_{1,t}, u_{1,f}, \ldots, u_{n,t}, u_{n,f}\}$. Since $u = (A, a^n)$ and every u_i is of the form (B, a) for some B, we have $m = n$. Moreover, since $u_1 \cdots u_{i-1}$ is of the form (C, a^{i-1}) for $1 \leq i \leq n$, we must have $u_i \in \{u_{i,f}, u_{i,t}\}$, because otherwise we would obtain $a^{-1} \in A$ (if $u_i \in \{u_{j,f}, u_{j,t}\}$ for $j > i$) or $a^{k+1} \in A$ (if $u_i \in \{u_{j,f}, u_{j,t}\}$ for $j < i$). Now we can define a truth assignment $\sigma : \{x_1, \ldots, x_n\} \to \{\text{true}, \text{false}\}$ as follows: $\sigma(x_i) = $ true if $u_i = u_{i,t}$ and $\sigma(x_i) = $ false if $u_i = u_{i,f}$. Since $a^jb \in A$ for every $1 \leq j \leq m$, it follows that for every $1 \leq j \leq m$ there is an $1 \leq i \leq n$ such that either $j \in P_i$ and $u_i = u_{i,t}$ (i.e., $\sigma(x_i) = $ true) or $j \in N_i$ and $u_i = u_{i,f}$ (i.e., $\sigma(x_i) = $ false). Thus, σ satisfies Ψ. □

The NP upper bound in Theorem 3 generalizes to the uniform case, where the independence relation I is part of the input but the number of cliques in a clique covering for $D = I \setminus (\Sigma \times \Sigma)$ is fixed by a constant. If we give up this restriction, the complexity goes up to PSPACE-completeness:

Theorem 5. *The following problem is* PSPACE*-complete:*
INPUT: An independence relation $I \subseteq \Sigma \times \Sigma$ and words $u, u_1, \ldots, u_n \in \Gamma^*$
QUESTION: $u \in \{u_1, \ldots, u_n\}^*$ in $\mathrm{FIM}(\Sigma, I)$?

5 $\mathrm{FIM}(\Sigma, I)$ Modulo an Idempotent Presentation

Let $I \subseteq \Sigma \times \Sigma$ be an independence relation. An *idempotent presentation* over (Σ, I) is a finite set of identities $P = \{(e_i, f_i) \mid 1 \leq i \leq n\}$, where every e_i and f_i is an idempotent element in $\mathrm{FIM}(\Sigma, I)$. Based on a reduction to Rabin's tree theorem, Margolis and Meakin have shown that for $I = \emptyset$, the uniform word problem for quotient monoids of the form $\mathrm{FIM}(\Sigma)/P$ (with P idempotent) is decidable [10]. Here, "uniform" means that the idempotent presentation P is part of the input. Recently, in [8] it was shown that the uniform word problem is EXPTIME-complete and that for every fixed idempotent presentation P the word problem for $\mathrm{FIM}(\Sigma)/P$ can be solved in logspace.

In this section we prove that the uniform word problem for monoids of the form $\mathrm{FIM}(\Sigma, I)/P$ (with P an idempotent presentation over (Σ, I)) is decidable if and only if the dependence relation $D = (\Sigma \times \Sigma) \setminus I$ is transitive, and for the transitive case we prove EXPTIME-completeness. Clearly, EXPTIME-hardness follows directly from [8]. For the upper bound, we use analogously to [10] a closure operation on subsets of $G(\Sigma, I)$. Assume that P is an idempotent presentation. Consider a pair $(e, f) \in P$. Then we have $e = (E, 1)$ and $f = (F, 1)$, where E and F are finite and closed subsets of the graph group $G(\Sigma, I)$ and $1 \in E \cap F$. In the following, we identify the pair $(E, 1)$ with the finite closed set E. Since e and f are idempotents of $\mathrm{FIM}(\Sigma, I)$, we can replace the relation $e = f$ by the two relations $e = ef$ and $f = ef$ without changing the quotient monoid [10]. Hence, for every pair $(E, F) \in P$, we can assume $E \subseteq F$.

Now assume that $A, B \subseteq G(\Sigma, I)$ are finite and closed. We write $A \Rightarrow_P B$ if and only if there exists $(E, F) \in P$ (hence $E \subseteq F$) and $f \in G(\Sigma, I)$ such that $fE \subseteq A$ and $B = \overline{A \cup fF}$. It is easy to see that the relation $\Rightarrow_P$ is strongly confluent, i.e., if $A \Rightarrow_P B$ and $A \Rightarrow_P C$, then there exists D such that $B \Rightarrow_P D$ and $C \Rightarrow_P D$. Hence, $A \overset{*}{\Leftrightarrow}_P B$ if and only if there exists C such that $A \overset{*}{\Rightarrow}_P C$ and $B \overset{*}{\Rightarrow}_P C$. Define $\mathrm{cl}_P(A) = \bigcup\{B \subseteq G(\Sigma, I) \mid A \overset{*}{\Rightarrow}_P B\} \subseteq G(\Sigma, P)$.

Lemma 2. *Let $(A, g), (B, h) \in \mathrm{FIM}(\Sigma, I)$. Then $(A, g) = (B, h)$ in $\mathrm{FIM}(\Sigma, I)/P$ if and only if $g = h$ in $G(\Sigma, I)$ and $\mathrm{cl}_P(A) = \mathrm{cl}_P(B)$.*

Using Lemma 2, we can generalize the EXPTIME upper bound from [8] for the case $I = \emptyset$: If $I \subseteq \Sigma \times \Sigma$ is an independence relation with $D = (\Sigma \times \Sigma) \setminus I$ transitive, then Σ is a disjoint union of D-cliques $\Sigma_1, \ldots, \Sigma_n$. Hence, $\mathrm{FIM}(\Sigma, I)$ is the direct product $\prod_{i=1}^n \mathrm{FIM}(\Sigma_i)$ of free inverse monoids. Moreover, a closed set $A \subseteq G(\Sigma, I)$ is a direct product $A = \prod_{i=1}^n A_i$ with $A_i \subseteq F(\Sigma_i)$ closed. By embedding each of the free groups $F(\Sigma_i)$ into a free group $F(\Theta)$ for a sufficiently large alphabet Θ, we can represent

A by an n-tuple $(A_1, \ldots, A_n)$ of closed subsets $A_i \subseteq F(\Theta)$. Computing the closure $\mathrm{cl}_P(A)$ corresponds to computing the *simultaneous fixpoint* of a monotonic mapping on $F(\Theta)^n$. Analogously to [8], we can formalize this fixpoint computation in the modal μ-calculus, interpreted over the Cayley graph of $F(\Theta)$. But for this, we need in contrast to [8] the modal μ-calculus with *simultaneous fixpoints*. On the other hand, the latter logic can be translated into the "ordinary" modal μ-calculus [1] without increasing the complexity of the model-checking problem. Since the model-checking problem of the modal μ-calculus over context-free graphs (which include Cayley graphs of free groups) belongs to EXPTIME [7,18], we finally get:

Theorem 6. *The following problem is* EXPTIME*-complete:*
INPUT: An independence relation $I \subseteq \Sigma \times \Sigma$ with $(\Sigma \times \Sigma) \setminus I$ transitive, an idempotent presentation P over (Σ, I) and words $u, v \in \Gamma^*$.
QUESTION: $u = v$ in $\mathrm{FIM}(\Sigma, I)/P$*?*

For a fixed idempotent presentation, we can again generalize a corresponding result from [8]:

Theorem 7. *If $I \subseteq \Sigma \times \Sigma$ is an independence relation with $(\Sigma \times \Sigma) \setminus I$ transitive and P is an idempotent presentation over (Σ, I), then the word problem for* $\mathrm{FIM}(\Sigma, I)/P$ *can be solved in (i) linear time on a RAM and (ii) logspace on a Turing machine.*

Theorem 6 and 7 are an interesting contrast to a result from [12] stating that the variety of E-unitary inverse monoids over an Abelian cover has an undecidable word problem. The difference is that in our setting pairs we only consider pairs (A, g), where A has to be closed, whereas in [12] this restriction is not imposed.

For a non-transitive dependence relation D we can encode the acceptance problem for a Turing-machine T in the word problem for $\mathrm{FIM}(\Sigma, I)/P$. Let $\Sigma = \{a, b, c\}$ and assume that $(a, c), (b, c) \in D$ but $(a, b) \in I$. Then a and b generate in the Cayley graph of $G(\{a, b, c\}, I)$ a two dimensional grid. Using the letter c, which is dependent from both a and b, we can encode a labelling of the grid-points with tape symbols and states of T. With the rewrite relation $\Rightarrow_P$ we generate a labelling consistent with the transition function of T and the input of T. Hence, we have:

Theorem 8. *Let $I \subseteq \Sigma \times \Sigma$ be an independence relation with $D = (\Sigma \times \Sigma) \setminus I$* not *transitive. Then there exists an idempotent presentation P over (Σ, I) such that the word problem for* $\mathrm{FIM}(\Sigma, I)/P$ *is undecidable.*

For the generalized word problem of $\mathrm{FIM}(\Sigma, I)/P$, we can prove undecidability even for a transitive dependence relation: Let $\Sigma = \{a, b, c, d\}$, $I = \{a, b\} \times \{c, d\} \cup \{c, d\} \times \{a, b\}$, and let the idempotent presentation P contain all identities $\alpha\alpha^{-1} = 1$ for $\alpha \in \Gamma$. Then $\mathrm{FIM}(\Sigma, I)/P$ is a direct product of two free groups of rank 2. By [13], this group has an undecidable generalized word problem. The only remaining case is a dependence relation, which consists of one clique of arbitrary size together with isolated nodes. The corresponding free partially commutative group is of the form $F \times \mathbb{Z}^k$, where F is a free group of arbitrary rank. It remains open, whether for such a dependence relation, the generalized word problem for $\mathrm{FIM}(\Sigma, I)/P$ is decidable for every idempotent presentation P. For the group $F \times \mathbb{Z}^k$ the generalized word problem is decidable [6].

References

1. H. Bekic. Definable operation in general algebras, and the theory of automata and flowcharts. In *Programming Languages and Their Definition*, LNCS 177, pages 30–55. Springer, 1984.
2. R. Cori and D. Perrin. Automates et commutations partielles. *RAIRO — Inform. Théor. Appl.*, 19:21–32, 1985.
3. V. Diekert. *Combinatorics on Traces*. LNCS 454. Springer, 1990.
4. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.
5. C. Droms. Graph groups, coherence and three-manifolds. *J. Algebra*, 106(2):484–489, 1985.
6. I. Kapovich, R. Weidmann, and A. Myasnikov. Foldings, graphs of groups and the membership problem. *Internat. J. Algebra Comput.*, 15(1):95–128, 2005.
7. O. Kupferman and M. Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proc. CAV 2000*, LNCS 1855, pages 36–52. Springer, 2000.
8. M. Lohrey and N. Ondrusch. Inverse monoids: decidability and complexity of algebraic questions. submitted, short version appeared in *Proc. MFCS 2005*, LNCS 3618, pages 664–675. Springer, 2005.
9. S. Margolis and J. Meakin. E-unitary inverse monoids and the Cayley graph of a group presentation. *J. Pure Appl. Algebra*, 58(1):45–76, 1989.
10. S. Margolis and J. Meakin. Inverse monoids, trees, and context-free languages. *Trans. Amer. Math. Soc.*, 335(1):259–276, 1993.
11. S. Margolis, J. Meakin, and M. Sapir. Algorithmic problems in groups, semigroups and inverse semigroups. In *Semigroups, Formal Languages and Groups*, pages 147–214. Kluwer, 1995.
12. J. Meakin and M. Sapir. The word problem in the variety of inverse semigroups with Abelian covers. *J. London Math. Soc. (2)*, 53(1):79–98, 1996.
13. K. A. Mihailova. The occurrence problem for direct products of groups. *Math. USSR Sbornik*, 70:241–251, 1966. English translation.
14. W. Munn. Free inverse semigroups. *Proc. London Math. Soc.*, 30:385–404, 1974.
15. M. Petrich. *Inverse semigroups*. Wiley, 1984.
16. J. Stephen. Presentations of inverse monoids. *J. Pure Appl. Algebra*, 63:81–112, 1990.
17. A. Veloso da Costa. *Γ-Produtos de Monóides e Semigrupos*. PhD thesis, Universidade do Porto, 2003.
18. I. Walukiewicz. Pushdown processes: games and model-checking. *Inform. and Comput.*, 164(2):234–263, 2001.
19. C. Wrathall. The word problem for free partially commutative groups. *J. Symbolic Comput.*, 6(1):99–104, 1988.

Learning Bayesian Networks Does Not Have to Be NP-Hard

Norbert Dojer

Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warszawa, Poland
dojer@mimuw.edu.pl

Abstract. We propose an algorithm for learning an optimal Bayesian network from data. Our method is addressed to biological applications, where usually datasets are small but sets of random variables are large. Moreover we assume that there is no need to examine the acyclicity of the graph.

We provide polynomial bounds (with respect to the number of random variables) for time complexity of our algorithm for two generally used scoring criteria: Minimal Description Length and Bayesian-Dirichlet equivalence.

1 Introduction

The framework of Bayesian networks is widely used in computational molecular biology. In particular, it appears attractive in the field of inferring a gene regulatory network structure from microarray expression data.

Researchers dealing with Bayesian networks have generally accepted that without restrictive assumptions, learning Bayesian networks from data is NP-hard with respect to the number of network vertices. This belief originates from a number of discouraging complexity results, which emerged over the last few years. Chickering [1] shows that learning an optimal network structure is NP-hard for the BDe scoring criterion with an arbitrary prior distribution, even when each node has at most two parents and the dataset is trivial. In this approach a reduction of a known NP-complete problem is based on constructing a complicated prior distribution for the BDe score. Chickering et al. [2] show that learning an optimal network structure is NP-hard for large datasets, even when each node has at most three parents and a scoring criterion favors the simplest model able to represent the distribution underlying the dataset exactly (as most of scores used in practice do). By a large dataset the authors mean the one which reflects the underlying distribution exactly. Moreover, this distribution is assumed to be directly accessible, so the complexity of scanning the dataset does not influence the result. In other papers [3,4], NP-hardness of the problem of learning Bayesian networks from data with some constraints on the structure of a network is shown.

On the other hand, the known algorithms allow to learn the structure of optimal networks having up to 20-40 vertices [5]. Consequently, a large amount of work has been dedicated to heuristic search techniques of identifying good models [6,7].

In the present paper we propose a new algorithm for learning an optimal network structure. Our method is addressed to the case when a dataset is small and there is no need to examine the acyclicity of the graph. The first condition does not bother in biological applications, because expenses of microarray experiments restrict the amount of expression data. The second assumption is satisfied in many cases, e.g. when working with dynamic Bayesian networks or when the sets of regulators and regulatees are disjoint.

We investigate the worst-case time complexity of our algorithm for generally used scoring criteria. We show that the algorithm works in polynomial time for both MDL and BDe scores. Experiments with real biological data show that, even for large floral genomes, the algorithm for the MDL score works in a reasonable time.

2 Algorithm

A *Bayesian network* (BN) $\mathcal{N}$ is a representation of a joint distribution of a set of discrete random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. The representation consists of two components:

- a directed acyclic graph $\mathcal{G} = (\mathbf{X}, \mathbf{E})$ encoding conditional (in-)dependencies
- a family θ of conditional distributions $P(X_i|\mathbf{Pa}_i)$, where

$$\mathbf{Pa}_i = \{Y \in \mathbf{X} | (Y, X_i) \in \mathbf{E}\}$$

The joint distribution of $\mathbf{X}$ is given by

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}_i) \tag{1}$$

The problem of learning a BN is understood as follows: given a multiset of $\mathbf{X}$-instances $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ find a network graph $\mathcal{G}$ that best matches $\mathcal{D}$. The notion of a good match is formalized by means of a *scoring function* $S(\mathcal{G} : \mathcal{D})$ having positive values and minimized for the best matching network. Thus the point is to find a directed acyclic graph $\mathcal{G}$ with the set of vertices $\mathbf{X}$ minimizing $S(\mathcal{G} : \mathcal{D})$.

In the present paper we consider the case when there is no need to examine the acyclicity of the graph, for example:

- When edges must be directed according to a given partial order, in particular when the sets of potential regulators and regulatees are disjoint.
- When dealing with *dynamic* Bayesian networks. A dynamic BN describes stochastic evolution of a set of random variables over discretized time. Therefore conditional distributions refer to random variables in neighboring time points. The acyclicity constraint is relaxed, because the "unrolled" graph

(with a copy of each variable in each time point) is always acyclic (see [7,8] for more details). The following considerations apply to dynamic BNs as well.

In the sequel we consider some assumptions on the form of a scoring function. The first one states that $S(\mathcal{G} : \mathcal{D})$ decomposes into a sum over the set of random variables of *local scores*, depending on the values of a variable and its parents in the graph only.

Assumption 1 (additivity). $S(\mathcal{G} : \mathcal{D}) = \sum_{i=1}^{n} s(X_i, \mathbf{Pa}_i : \mathcal{D}|_{\{X_i\}\cup\mathbf{Pa}_i})$, *where* $\mathcal{D}|_{\mathbf{Y}}$ *denotes the restriction of* $\mathcal{D}$ *to the values of the members of* $\mathbf{Y} \subseteq \mathbf{X}$.

When there is no need to examine the acyclicity of the graph, this assumption allows to compute the parents set of each variable independently. Thus the point is to find $\mathbf{Pa}_i$ minimizing $s(X_i, \mathbf{Pa}_i : \mathcal{D}|_{\{X_i\}\cup\mathbf{Pa}_i})$ for each i.

Let us fix a dataset $\mathcal{D}$ and a random variable X. We denote by $\mathbf{X}'$ the set of potential parents of X (possibly smaller than $\mathbf{X}$ due to given constraints on the structure of the network). To simplify the notation we continue to write $s(\mathbf{Pa})$ for $s(X, \mathbf{Pa} : \mathcal{D}|_{\{X\}\cup\mathbf{Pa}})$.

The following assumption expresses the fact that scoring functions decompose into 2 components: g penalizing the complexity of a network and d evaluating the possibility of explaining data by a network.

Assumption 2 (splitting). $s(\mathbf{Pa}) = g(\mathbf{Pa}) + d(\mathbf{Pa})$ *for some functions* $g, d : \mathcal{P}(\mathbf{X}) \to \mathbb{R}^+$ *satisfying* $\mathbf{Pa} \subseteq \mathbf{Pa}' \Longrightarrow g(\mathbf{Pa}) \leq g(\mathbf{Pa}')$.

This assumption is used in the following algorithm to avoid considering networks with inadequately large component g.

Algorithm 1.

```
1. $\mathbf{Pa} := \emptyset$
2. for each $\mathbf{P} \subseteq \mathbf{X}'$ chosen according to $g(\mathbf{P})$
   (a) if $s(\mathbf{P}) < s(\mathbf{Pa})$ then $\mathbf{Pa} := \mathbf{P}$
   (b) if $g(\mathbf{P}) \geq s(\mathbf{Pa})$ then return $\mathbf{Pa}$; stop
```

In the above algorithm *choosing according to* $g(\mathbf{P})$ means choosing increasingly with respect to the value of the component g of the local score.

Theorem 1. *Suppose that the scoring function satisfies Assumptions 1-2. Then Algorithm 1 applied to each random variable finds an optimal network.*

Proof. The theorem follows from the fact that all the potential parents sets $\mathbf{P}$ omitted by the algorithm satisfy $s(\mathbf{P}) \geq g(\mathbf{P}) \geq s(\mathbf{Pa})$, where $\mathbf{Pa}$ is the returned set.

A disadvantage of the above algorithm is that finding a proper subset $\mathbf{P} \subseteq \mathbf{X}'$ involves computing $g(\mathbf{P}')$ for all $\subseteq$-successors $\mathbf{P}'$ of previously chosen subsets. It may be avoided when a further assumption is imposed.

Assumption 3 (uniformity). $|\mathbf{Pa}| = |\mathbf{Pa}'| \Longrightarrow g(\mathbf{Pa}) = g(\mathbf{Pa}')$.

The above assumption suggests the notation $\hat{g}(|\mathbf{Pa}|) = g(\mathbf{Pa})$. The following algorithm uses the uniformity of g to reduce the number of computations of the component g.

Algorithm 2.

```
1. $\mathbf{Pa} := \emptyset$
2. for $p = 1$ to $n$
   (a) if $\hat{g}(p) \geq s(\mathbf{Pa})$ then return $\mathbf{Pa}$; stop
   (b) $\mathbf{P} = argmin_{\{\mathbf{Y} \subseteq \mathbf{X}' : |\mathbf{Y}| = p\}} s(\mathbf{Y})$
   (c) if $s(\mathbf{P}) < s(\mathbf{Pa})$ then $\mathbf{Pa} := \mathbf{P}$
```

Theorem 2. *Suppose that the scoring function satisfies Assumptions 1-3. Then Algorithm 2 applied to each random variable finds an optimal network.*

Proof. The theorem follows from the fact that all the potential parents sets $\mathbf{P}$ omitted by the algorithm satisfy $s(\mathbf{P}) \geq \hat{g}(|\mathbf{P}|) \geq s(\mathbf{Pa})$, where $\mathbf{Pa}$ is the returned set.

The following sections are devoted to two generally used scoring criteria: Minimal Description Length and Bayesian-Dirichlet equivalence. We consider possible decompositions of the local scores and analyze the computational cost of the above algorithms.

3 Minimal Description Length

The Minimal Description Length (MDL) scoring criterion originates from information theory [9]. A network $\mathcal{N}$ is viewed here as a model of compression of a dataset $\mathcal{D}$. The optimal model minimizes the total length of the description, i.e. the sum of the description length of the model and of the compressed data.

Let us fix a dataset $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and a random variable X. Recall the decomposition $s(\mathbf{Pa}) = g(\mathbf{Pa}) + d(\mathbf{Pa})$ of the local score for X. In the MDL score $g(\mathbf{Pa})$ stands for the length of the description of the local part of the network (i.e. the edges ingoing to X and the conditional distribution $P(X|\mathbf{Pa})$) and $d(\mathbf{Pa})$ is the length of the compressed version of X-values in $\mathcal{D}$.

Let k_Y denote the cardinality of the set $\mathcal{V}_Y$ of possible values of the random variable $Y \in \mathbf{X}$. Thus we have

$$g(\mathbf{Pa}) = |\mathbf{Pa}| \log n + \frac{\log N}{2}(k_X - 1) \prod_{Y \in \mathbf{Pa}} k_Y$$

where $\frac{\log N}{2}$ is the number of bits we use for each numeric parameter of the conditional distribution. This formula satisfies Assumption 2 but fails to satisfy Assumption 3. Therefore Algorithm 1 can be used to learn an optimal network, but Algorithm 2 cannot.

However, for many applications we may assume that all the random variables attain values from the same set $\mathcal{V}$ of cardinality k. In this case we obtain the formula

$$g(\mathbf{Pa}) = |\mathbf{Pa}| \log n + \frac{\log N}{2}(k-1)k^{|\mathbf{Pa}|}$$

which satisfies Assumption 3. For simplicity, we continue to work under this assumption. The general case may be handled in much the same way.

Compression with respect to the network model is understood as follows: when encoding the X-values, the values of **Pa**-instances are assumed to be known. Thus the optimal encoding length is given by

$$d(\mathbf{Pa}) = N \cdot H(X|\mathbf{Pa})$$

where $H(X|\mathbf{Pa}) = -\sum_{v \in \mathcal{V}} \sum_{\mathbf{v} \in \mathcal{V}^{\mathbf{Pa}}} P(v, \mathbf{v}) \log P(v|\mathbf{v})$ is the conditional entropy of X given **Pa** (the distributions are estimated from $\mathcal{D}$).

Since all the assumptions from the previous section are satisfied, Algorithm 2 may be applied to learn the optimal network. Let us turn to the analysis of its complexity.

Theorem 3. *The worst-case time complexity of Algorithm 2 for the MDL score is* $\mathcal{O}(n^{\log_k N} N \log_k N)$.

Proof. The proof consists in finding a number p satisfying $\hat{g}(p) \geq s(\emptyset)$.

Given $v \in \mathcal{V}$, we denote by N_v the number of samples in $\mathcal{D}$ with $X = v$. By Jensen's Theorem we have

$$d(\emptyset) = N \cdot H(X|\emptyset) = N \sum_{v \in \mathcal{V}} \frac{N_v}{N} \log \frac{N}{N_v} \leq N \log \sum_{v \in \mathcal{V}} 1 = N \log k$$

Hence $s(\emptyset) \leq N \log k + (k-1)\frac{\log N}{2}$ and it suffices to find p satisfying

$$p \log n + k^p (k-1) \frac{\log N}{2} \geq N \log k + (k-1) \frac{\log N}{2}$$

It is easily seen that the above assertion holds for $p \geq \log_k N$ whenever $N > 5$.

Therefore we do not have to consider subsets with more than $\log_k N$ parent variables. Since there are $\mathcal{O}(n^p)$ subsets with at most p parents and a subset with p elements may be examined in pN steps, the total complexity of the algorithm is $\mathcal{O}(n^{\log_k N} N \log_k N)$. $\square$

4 Bayesian-Dirichlet Equivalence

The Bayesian-Dirichlet equivalence (BDe) scoring criterion originates from Bayesian statistics [10]. Given a dataset $\mathcal{D}$ the optimal network structure $\mathcal{G}$ maximizes the *posterior* conditional probability $P(\mathcal{G}|\mathcal{D})$. We have

$$P(\mathcal{G}|\mathcal{D}) \propto P(\mathcal{G})P(\mathcal{D}|\mathcal{G}) = P(\mathcal{G})\int P(\mathcal{D}|\mathcal{G},\theta)P(\theta|\mathcal{G})d\theta$$

where $P(\mathcal{G})$ and $P(\theta|\mathcal{G})$ are *prior* probability distributions on graph structures and conditional distributions' parameters, respectively, and $P(\mathcal{D}|\mathcal{G},\theta)$ is evaluated due to (1).

Heckerman et al. [6], following Cooper and Herskovits [10], identified a set of independence assumptions making possible decomposition of the integral in the above formula into a product over $\mathbf{X}$. Under this condition, together with a similar one regarding decomposition of $P(\mathcal{G})$, the scoring criterion

$$S(\mathcal{G}:\mathcal{D}) = -\log P(\mathcal{G}) - \log P(\mathcal{D}|\mathcal{G})$$

obtained by taking $-\log$ of the above term satisfies Assumption 1. Moreover, the decomposition $s(\mathbf{Pa}) = g(\mathbf{Pa}) + d(\mathbf{Pa})$ of the local scores appears as well, with the components g and d derived from $-\log P(\mathcal{G})$ and $-\log P(\mathcal{D}|\mathcal{G})$, respectively.

The distribution $P((\mathbf{X},\mathbf{E})) \propto \alpha^{|\mathbf{E}|}$ with a penalty parameter $0 < \alpha < 1$ in general is used as a prior over the network structures. This choice results in the function

$$g(|\mathbf{Pa}|) = |\mathbf{Pa}|\log\alpha^{-1}$$

satisfying Assumptions 2 and 3.

However, it should be noticed that there are also used priors which satisfy neither Assumption 2 nor 3, e.g. $P(\mathcal{G}) \propto \alpha^{\Delta(\mathcal{G},\mathcal{G}_0)}$, where $\Delta(\mathcal{G},\mathcal{G}_0)$ is the cardinality of the symmetric difference between the sets of edges in $\mathcal{G}$ and in the prior network $\mathcal{G}_0$.

The *Dirichlet distribution* is generally used as a prior over the conditional distributions' parameters. It yields

$$d(\mathbf{Pa}) = \log\left(\prod_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \frac{\Gamma(\sum_{v\in\mathcal{V}}(H_{v,\mathbf{v}}+N_{v,\mathbf{v}}))}{\Gamma(\sum_{v\in\mathcal{V}} H_{v,\mathbf{v}})} \prod_{v\in\mathcal{V}} \frac{\Gamma(H_{v,\mathbf{v}})}{\Gamma(H_{v,\mathbf{v}}+N_{v,\mathbf{v}})}\right)$$

where Γ is the *Gamma* function, $N_{v,\mathbf{v}}$ denotes the number of samples in $\mathcal{D}$ with $X = v$ and $\mathbf{Pa} = \mathbf{v}$, and $H_{v,\mathbf{v}}$ is the corresponding *hyperparameter* of the Dirichlet distribution.

Setting all the hyperparameters to 1 yields

$$d(\mathbf{Pa}) = \log\left(\prod_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \frac{(k-1+\sum_{v\in\mathcal{V}} N_{v,\mathbf{v}})!}{(k-1)!} \prod_{v\in\mathcal{V}} \frac{1}{N_{v,\mathbf{v}}!}\right) =$$
$$= \sum_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \left(\log(k-1+\sum_{v\in\mathcal{V}} N_{v,\mathbf{v}})! - \log(k-1)! - \sum_{v\in\mathcal{V}} \log N_{v,\mathbf{v}}!\right)$$

where $k = |\mathcal{V}|$. For simplicity, we continue to work under this assumption. The general case may be handled in a similar way.

The following result allows to refine the decomposition of the local score into the sum of the components g and d.

Proposition 1. *Define* $d_{min} = \sum_{v\in\mathcal{V}} (\log(k-1+N_v)! - \log(k-1)! - \log N_v!)$, *where* N_v *denotes the number of samples in* $\mathcal{D}$ *with* $X = v$. *Then* $d(\mathbf{Pa}) \geq d_{min}$ *for each* $\mathbf{Pa} \in \mathbf{X}$.

Proof. Fix $\mathbf{Pa} \subseteq \mathbf{X}$. Given $\mathbf{v} \in \mathcal{V}^{|\mathbf{Pa}|}$, we denote by $N_\mathbf{v}$ the number of samples in $\mathcal{D}$ with $\mathbf{Pa} = \mathbf{v}$. We have

$$
\begin{aligned}
d(\mathbf{Pa}) &= \sum_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \left(\log(k-1+N_\mathbf{v})! - \log(k-1)! - \sum_{v\in\mathcal{V}} \log N_{v,\mathbf{v}}!\right) = \\
&= \sum_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \left(\sum_{i=k}^{k-1+N_\mathbf{v}} \log i - \sum_{v\in\mathcal{V}} \sum_{i=1}^{N_{v,\mathbf{v}}} \log i\right) \geq \\
&\geq \sum_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \sum_{v\in\mathcal{V}} \left(\sum_{i=k}^{k-1+N_{v,\mathbf{v}}} \log i - \sum_{i=1}^{N_{v,\mathbf{v}}} \log i\right) = \sum_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \sum_{v\in\mathcal{V}} \sum_{i=1}^{N_{v,\mathbf{v}}} \log \frac{k-1+i}{i} = \\
&= \sum_{v\in\mathcal{V}} \sum_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \sum_{i=1}^{N_{v,\mathbf{v}}} \log(\frac{k-1}{i}+1) \geq \sum_{v\in\mathcal{V}} \sum_{i=1}^{N_v} \log(\frac{k-1}{i}+1) = \\
&= \sum_{v\in\mathcal{V}} \sum_{i=1}^{N_v} \log \frac{k-1+i}{i} = \sum_{v\in\mathcal{V}} \left(\sum_{i=k}^{k-1+N_v} \log i - \sum_{i=1}^{N_v} \log i\right) = \\
&= \sum_{v\in\mathcal{V}} (\log(k-1+N_v)! - \log(k-1)! - \log N_v!) = d_{min}
\end{aligned}
$$

The first inequality follows from the fact that given $N_\mathbf{v}$ the sum

$$\sum_{v\in\mathcal{V}} \sum_{i=k}^{k-1+N_{v,\mathbf{v}}} \log i$$

maximizes when $N_\mathbf{v} = N_{v,\mathbf{v}}$ for some $v \in \mathcal{V}$. Similarly, the second inequality holds, because given N_v the sum

$$\sum_{\mathbf{v}\in\mathcal{V}^{|\mathbf{Pa}|}} \sum_{i=1}^{N_{v,\mathbf{v}}} \log(\frac{k-1}{i}+1)$$

minimizes when $N_v = N_{v,\mathbf{v}}$ for some $\mathbf{v} \in \mathcal{V}^{|\mathbf{Pa}|}$. □

By the above proposition, the decomposition of the local score given by $s(\mathbf{Pa}) = g'(\mathbf{Pa}) + d'(\mathbf{Pa})$ with the components $g'(\mathbf{Pa}) = g(\mathbf{Pa}) + d_{min}$ and $d'(\mathbf{Pa}) = d(\mathbf{Pa}) - d_{min}$ satisfies all the assumptions required by Algorithm 2. Let us turn to the analysis of its complexity.

Theorem 4. *The worst-case time complexity of Algorithm 2 for the BDe score with the decomposition of the local score given by* $s(\mathbf{Pa}) = g'(\mathbf{Pa}) + d'(\mathbf{Pa})$ *is* $\mathcal{O}(n^{N \log_{\alpha^{-1}} k} N^2 \log_{\alpha^{-1}} k)$.

Proof. The proof consists in finding a number p satisfying $\hat{g}'(p) \geq s(\emptyset)$.

We have

$$\begin{aligned}
&d'(\emptyset) = d(\emptyset) - d_{min} = \\
&= \left(\log \frac{(k-1+N)!}{(k-1)!} - \sum_{v\in\mathcal{V}} \log N_v!\right) - \sum_{v\in\mathcal{V}} \left(\log \frac{(k-1+N_v)!}{(k-1)!} - \log N_v!\right) = \\
&= (\log(k-1+N)! - \log(k-1)!) - \sum_{v\in\mathcal{V}} (\log(k-1+N_v)! - \log(k-1)!) = \\
&= \sum_{i=k}^{k-1+N} \log i - \sum_{v\in\mathcal{V}} \sum_{i=k}^{k-1+N_v} \log i \leq \\
&\leq \sum_{i=k}^{k-1+N} \log i - \left(k \sum_{i=k}^{k-1+\lfloor\frac{N}{k}\rfloor} \log i + (N - k\lfloor\frac{N}{k}\rfloor) \log(k + \lfloor\frac{N}{k}\rfloor)\right) = \\
&= \log e \left(\sum_{i=k}^{k-1+N} \ln i - k\left(\sum_{i=k}^{k-1+\lfloor\frac{N}{k}\rfloor} \ln i + (\frac{N}{k} - \lfloor\frac{N}{k}\rfloor) \ln(k + \lfloor\frac{N}{k}\rfloor)\right)\right) \leq \\
&\leq \log e \left(\int_k^{k+N} \ln x\,dx - k\left(\int_{k-1}^{k-1+\lfloor\frac{N}{k}\rfloor} \ln x\,dx + \int_{k-1+\lfloor\frac{N}{k}\rfloor}^{k-1+\frac{N}{k}} \ln x\,dx\right)\right) = \\
&= \log e \Bigg((k+N)(\ln(k+N) - 1) - k(\ln k - 1) - \\
&- k\left((k-1+\frac{N}{k})(\ln(k-1+\frac{N}{k}) - 1) - (k-1)(\ln(k-1) - 1)\right)\Bigg) = \\
&= N \log k + N \log \frac{k+N}{k^2 - k + N} + \log \frac{(1+\frac{N}{k})^k}{(1+\frac{N}{k^2-k})^{k^2-k}} \leq N \log k
\end{aligned}$$

The first inequality follows from the fact that the sum $\sum_{v\in\mathcal{V}} \log N_v!$ minimizes when the values of N_v are uniformly distributed over v. The last inequality is due to the fact that $k \geq 2$, and consequently $k^2 - k \geq k$.

Thus $s(\emptyset) \leq N \log k + d_{min}$, so we need p satisfying $p \log \alpha^{-1} \geq N \log k$, i.e. $p \geq N \log_{\alpha^{-1}} k$.

Therefore we do not have to consider subsets with more than $N \log_{\alpha^{-1}} k$ parent variables. Since there are $\mathcal{O}(n^p)$ subsets with at most p parents and a subset with p elements may be examined in pN steps, the total complexity of the algorithm is $\mathcal{O}(n^{N \log_{\alpha^{-1}} k} N^2 \log_{\alpha^{-1}} k)$. □

5 Application to Biological Data

Microarray experiments measure simultaneously expression levels of thousands of genes. Learning Bayesian networks from microarray data aims at discovering gene regulatory dependencies.

We tested the time complexity of learning an optimal dynamic BN on microarray experiments data of *Arabidopsis thaliana* (∼23 000 genes). There were 20 samples of time series data discretized into 3 expression levels. We ran our program on a single Pentium 4 CPU 2.8 GHz.

The computation time was 48 hours for the MDL score and 170 hours for the BDe score with a parameter $\log \alpha^{-1} = 5$. The program failed to terminate in a reasonable time for the BDe score with a parameter $\log \alpha^{-1} = 3$, i.e $\alpha \approx 0.05$.

6 Conclusion

The aim of this work was to provide an effective algorithm for learning optimal Bayesian network from data, applicable to microarray expression measurements. We have proposed a method addressed to the case when a dataset is small and there is no need to examine the acyclicity of the graph. We have provided polynomial bounds (with respect to the number of random variables) for time complexity of our algorithm for two generally used scoring criteria: Minimal Description Length and Bayesian-Dirichlet equivalence.

An application to real biological data shows that our algorithm works with large floral genomes in a reasonable time for the MDL score but fails to terminate for the BDe score with a sensible value of the penalty parameter.

Acknowledgements

The author is grateful to Jerzy Tiuryn for comments on previous versions of this work and useful discussions related to the topic. I also thank Bartek Wilczyński for implementing and running our learning algorithm.

This research was founded by Polish State Committee for Scientific Research under grant no. 3T11F02128.

References

1. Chickering, D.M.: Learning Bayesian networks is NP-complete. In Fisher, D., Lenz, H.J., eds.: Learning from Data: Artificial Inteligence and Statistics V. Springer-Verlag (1996)
2. Chickering, D.M., Heckerman, D., Meek, C.: Large-sample learning of Bayesian networks is NP-hard. Journal of Machine Learning Research **5** (2004) 1287–1330
3. Dasgupta, S.: Learning polytrees. In: Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99), San Francisco, CA, Morgan Kaufmann Publishers (1999) 134–141
4. Meek, C.: Finding a path is harder than finding a tree. J. Artificial Intelligence Res. **15** (2001) 383–389 (electronic)
5. Ott, S., Imoto, S., Miyano, S.: Finding optimal models for small gene networks. Pac. Symp. Biocomput. (2004) 557–567
6. Heckerman, D., Geiger, D., Chickering, D.M.: Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning **20**(3) (1995) 197–243

7. Neapolitan, R.E.: Learning Bayesian Networks. Prentice Hall (2003)
8. Friedman, N., Murphy, K., Russell, S.: Learning the structure of dynamic probabilistic networks. In Cooper, G.F., Moral, S., eds.: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Inteligence. (1998) 139–147
9. Lam, W., Bacchus, F.: Learning Bayesian belief networks: An approach based on the MDL principle. Computational Intelligence **10**(3) (1994) 269–293
10. Cooper, G.F., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning **9** (1992) 309–347

Lower Bounds for the Transition Complexity of NFAs*
(Extended Abstract)

Michael Domaratzki[1] and Kai Salomaa[2]

[1] Jodrey School of Computer Science, Acadia University, Wolfville, Nova Scotia B4P 2R6, Canada
mike.domaratzki@acadiau.ca
[2] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. We construct regular languages L_n, $n \geq 1$, such that any NFA recognizing L_n needs $\Omega(\mathrm{nsc}(L_n) \cdot \sqrt{\mathrm{nsc}(L_n)})$ transitions where $\mathrm{nsc}(L_n)$ is the nondeterministic state complexity of L_n. Also, we study trade-offs between the number of states and the number of transitions of an NFA. We show that adding one additional state can result in significant reductions in the number of transitions and that there exist regular languages L_n, $n \geq 2$, where the transition minimal NFA for L_n has more than $c \cdot \mathrm{nsc}(L_n)$ states, for some constant $c > 1$.

1 Introduction

Recently there has been much work on descriptional complexity, or state complexity, of deterministic finite automata (DFAs, see Section 2 for definitions). The results are surveyed and more references can be found in [5,13,21,22]. The state complexity of nondeterministic finite automaton (NFA) operations has been investigated by Holzer and Kutrib [10,11]. The minimal DFA equivalent to an arbitrary DFA can be found efficiently, whereas Jiang and Ravikumar [16] have shown that minimization of NFAs is PSPACE-complete and Malcher [20] has shown that minimization remains NP-complete for classes of NFAs that are nearly deterministic. A technique for proving lower bounds for the state complexity of NFAs has been given using fooling sets, see Hromkovič [12] or Glaister and Shallit [4]. Gruber and Holzer [8] show that it is computationally hard to decide whether lower bounds given by the fooling set technique can be reached.

The number of transitions gives, in some sense, a more realistic measure for the size of an NFA than the number of states. In a worst-case comparison of nondeterministic state complexity and transition complexity it has been recently established by Gruber and Holzer [9] and by Kari [18] that there exist finite languages L_n, $n \geq 1$, such that any NFA for L_n needs $\Omega(\frac{\mathrm{nsc}(L_n)^2}{\log(\mathrm{nsc}(L_n))})$ transitions.

* Research supported, in part, by the Natural Sciences and Engineering Research Council of Canada.

Here $\mathrm{nsc}(L)$ is the nondeterministic state complexity of L. The above shows that the nondeterministic transition complexity of regular languages, as a function of the nondeterministic state complexity, can approach the quadratic upper bound. The lower bounds of [9,18] are proved using counting arguments and the results do not yield efficiently constructible languages having a corresponding lower bound for their transition complexity. A non-trivial transition complexity lower bound, albeit a weaker lower bound, for a concrete family of regular languages can be obtained from the work of Hromkovič and Schnitger [14].

Here we give an explicit construction of a family of regular languages L_n, $n \geq 1$, such that the nondeterministic state complexity of L_n is $O(n)$ and any NFA for the language L_n needs $\Omega(n \cdot \sqrt{n})$ transitions. The lower bound relies on nontrivial combinatorial properties, namely on the existence of projective planes of arbitrarily large degree [2,17]. When considering the number of transitions as a function of the nondeterministic state complexity, our lower bound is better than the one obtained in [14] for concrete regular languages over a fixed alphabet. It should be noted that the family of languages in [14] was designed for the purpose of maximizing the increase of transition complexity when converting an ε-NFA to an NFA without ε-transitions, as opposed to maximizing transition complexity as a function of nondeterministic state complexity.

Our lower bound applies to a specific family of regular languages. A topic for further research would be to extend the result as a more general purpose technique for establishing lower bounds for transition complexity, in the spirit of the methods for proving lower bounds for nondeterministic state complexity [4,12]. The proof of our lower bound result introduces techniques that may turn out to be useful for work in this direction.

We also study the trade-offs between nondeterministic state complexity and the number of transitions needed by an NFA. By the *strict transition complexity* of a language L we mean the number of transitions needed by any NFA for L with $\mathrm{nsc}(L)$ states. It turns out that already allowing one additional state can cause a drastic reduction in the number of transitions, that is, there are languages L_n such that the strict transition complexity of L_n is $\Omega(n^2)$ but if an NFA for L_n can use $\mathrm{nsc}(L_n) + 1$ states it needs only $O(n)$ transitions. We show that for each $k \geq 1$ there are languages that exhibit an analogous gap in transition complexity when comparing NFAs that allow, respectively, $k - 1$ and k additional states compared to the size of the state-minimal NFA.

To conclude, we mention other work on transition complexity. Upper and lower bounds for the number of transitions of an NFA equivalent to a given regular expression are obtained in [15,19]. There the lower bounds use variable size alphabets. Inapproximability results for minimizing the number of NFA transitions for a given regular language are established by Gramlich and Schnitger [6].

2 Preliminaries

For $n \geq 1$ denote $[n] = \{1, \ldots, n\}$. The set of prefixes (respectively, suffixes) of a word $w \in \Sigma^*$ is denoted $\mathrm{pref}(w)$ (respectively, $\mathrm{suf}(w)$). The length of $w \in \Sigma^*$ is

$|w|$. An NFA is denoted as $A = (\Sigma, Q, q_0, Q_F, \delta)$ where Σ is the input alphabet, Q is the set of states, $q_0 \in Q$ is the start state, $Q_F \subseteq Q$ is the set of accepting states and $\delta \subseteq Q \times \Sigma \times Q$ gives the set of transitions. A state $q \in Q$ is useful if q is reachable from the start state and some accepting state is reachable from q. It is well known that the set of useful states can be efficiently found [21] and, without loss of generality, we assume that the NFAs under consideration have only useful states. A DFA is an NFA with $|\{(q, a, q') \in \delta \ : \ q' \in Q\}| \leq 1$ for all $(q, a) \in Q \times \Sigma$.

The nondeterministic state complexity of a regular language L is the smallest number of states of any NFA recognizing L. It is denoted as $\text{nsc}(L)$. Generally, we consider the transition complexity of a language L as a function of $\text{nsc}(L)$.

Definition 2.1. *Let L be a regular language. The (nondeterministic)* transition complexity *of L, $\text{tc}(L)$, is the smallest number of transitions of any NFA that recognizes L.*

For $k \geq 0$, the k-strict transition complexity *of L, $\text{stc}_k(L)$, is the smallest number of transitions of any NFA A for L such that A has at most $\text{nsc}(L) + k$ states.*

The 0-strict transition complexity of L is simply called the *strict transition complexity* of L. For a regular language L, the strict transition complexity of L is the smallest number of transitions needed by any NFA for L that has $\text{nsc}(L)$ states. For any regular language L and $k \geq 0$,

$$\text{nsc}(L) + 1 \leq \text{tc}(L) \leq \text{stc}_{k+1}(L) \leq \text{stc}_k(L). \tag{1}$$

3 Non-linear Lower Bound for Transition Complexity

We construct a family of languages L_n, $n \geq 1$, such that any NFA for L_n requires provably $\Omega(\text{nsc}(L_n) \cdot \sqrt{\text{nsc}(L_n)})$ transitions. The intuitive idea of the construction is that we define languages where the corresponding NFAs consist of two "parts" that require a large number of "connections", and the language is designed in a way that prevents "funneling" large numbers of connections through a single state.

3.1 Definition of a Language Associated with a Binary Relation

We begin by defining a class of finite languages with particular properties. Each language is associated with a binary relation on $[n]$, $n \geq 1$.

For $w = a_1 \cdots a_s$, $s \geq 1$, $a_i \in \Sigma$, $i = 1, \ldots, s$, and $1 \leq i, j \leq s$ the following notations are used for the prefix of w of length i, $w[\rightarrow i] = a_1 a_2 \cdots a_i$, and the suffix of w beginning from the jth position of w, $w[j \leftarrow] = a_j a_{j+1} \cdots a_s$. Also for $1 \leq i, j \leq s$ denote

$$w[i, j] = \begin{cases} a_i a_{i+1} \cdots a_{i+j} \text{ if } i + j \leq s, \\ a_i a_{i+1} \cdots a_s, \text{otherwise.} \end{cases}$$

Above $w[i,j]$ is the subword of length $j+1$ starting at the ith symbol of w, if this subword is "inside" of w and otherwise $w[i,j]$ is the suffix of w starting at the ith position.

Let w_0, w_1, w_2, … be the enumeration of all non-empty words over alphabet $\{0,1\}$ in length-lexicographic order. We define the following infinite word over the alphabet $\{0,1,\#\}$:

$$\alpha = w_0\#w_1\#w_2\#\ldots \;=\; 0\#1\#00\#01\#10\#11\#000\#\ldots$$

Let α_r, $r \geq 1$, be the prefix of α of length r.

Consider a fixed $n \geq 1$ such that our binary relations will be defined as subsets of $[n] \times [n]$. We denote $m = n + \lceil \log n \rceil$.[1] The value of m depends on n. It is easy to verify that the following property holds for all $1 \leq i \leq n$:

$$\text{the subword } \alpha_m[i, \lceil \log m \rceil - 1] \text{ occurs in only one position in } \alpha_m. \tag{2}$$

For $1 \leq i \leq n$, we define

$$f(i) = \alpha_m[i, \lceil \log m \rceil - 1]. \tag{3}$$

Here $f(i)$ is the subword of α_m of length $\lceil \log m \rceil$ beginning at position i. By (2), $f(i)$ occurs only once in the word α_m.

In the following of this section let $\Sigma = \{0,1,\#,\$\}$. Let $\omega_n \subseteq [n] \times [n]$ be an arbitrary binary relation. Now we define the language $L(\omega_n) \subseteq \Sigma^*$ as follows:

$$L(\omega_n) = \{\ \alpha_m[\rightarrow i + \lceil \log m \rceil - 1]\ \$\ \alpha_m[j \leftarrow]\ \mid\ (i,j) \in \omega_n,\ 1 \leq i,j \leq n\ \}.$$

Intuitively, the language $L(\omega_n)$ consists of, for each $(i,j) \in \omega_n$, all words beginning with the prefix of α_m that ends with the unique occurrence of the subword $f(i)$, followed by the middle marker \$ and the suffix of α_m beginning with the unique occurrence of the subword $f(j)$.

Since $m \leq 2n$, the following lemma is immediate.

Lemma 3.1. *Let ω_n be any binary relation on $[n]$. Then* $\mathrm{nsc}(L(\omega_n)) \in O(n)$.

□

Next we define some terminology associated with an arbitrary NFA recognizing some of the languages defined above. We use graph theoretic arguments and below we refer to transitions and states of the NFA interchangeably as edges and nodes, respectively, of the corresponding state graph.

Let $A = (\Sigma, Q, q_0, Q_F, \delta)$ be an NFA recognizing $L(\omega_n)$ and $k \geq 1$. We say that a state q_2 is *k-reachable* from a state q_1 (via word w) if, for $w \in \Sigma^k$, $q_2 \in \delta(q_1, w)$. Note that k-reachability refers to reachability using a word of length exactly k.

By a *funnel edge* (or transition) we mean any edge e of the underlying graph of A labeled by \$. A *funnel* of A consists of a funnel edge e together with

[1] By using endmarkers, it would be possible to use words of length exactly n. The additive term $\lceil \log n \rceil$ will not change bounds that ignore constant factors, and it will make parts of the notation more uniform. All our logarithms are to the base 2.

– all states $q \in Q$ such that the in-node of e is $\lceil \log m \rceil$-reachable from q, these are called the *in-states* of the funnel associated with e; and,
– all states $q \in Q$ such that q is $\lceil \log m \rceil$-reachable from the out-node of e, these are called the *out-states* of the funnel associated with e.

Below we establish some properties of A. Without loss of generality we can assume that A has a single accepting state, $Q_F = \{q_f\}$. This follows from the observation that any word of $L(\omega_n)$ ends with a unique suffix of length $\lceil \log m \rceil$.

For $1 \leq i \leq n$ we define the following subsets of Q:

$$B(i) = \delta(q_0, \alpha_m[\to (i-1)]),$$
$$C(i) = \{q \in Q \mid q_f \in \delta(q, \alpha_m[(i + \lceil \log m \rceil) \leftarrow])\}.$$

The set $B(i)$ consists of states that are reachable from the start state using a prefix of length $i-1$ of some word in $L(\omega_n)$ that does not contain the middle marker \$ (the words of $L(\omega_n)$ have only one possible prefix without the marker \$ having length $i-1$, but it is not the prefix of all words in $L(\omega_n)$). The set $C(i)$ consists of states that can accept a suffix of α_m starting from the $(i+\lceil \log m \rceil)$th position of α_m.

The sets $B(i)$ and $B(j)$ (respectively, $C(i)$ and $C(j)$), $i \neq j$, may in general contain common elements. However, the crucial property that will be used in our lower bound estimate is that if a state q belongs to some funnel then q can belong to at most one of the sets $B(i)$, $1 \leq i \leq n$, (respectively, at most one of the sets $C(i)$, $1 \leq i \leq n$). This property will be stated in (4) and (5) below.

The below properties (F1)–(F3) follow from (2) and from the definition of the words $f(i)$, $1 \leq i \leq n$.

(F1) If a state $q \in B(i)$, $1 \leq i \leq n$, is in the funnel corresponding to edge e, then the in-node of e is reachable from q via the word $f(i)$ (as in (3)), and $f(i)$ is the only word of length $\lceil \log m \rceil$ with this property.
(F2) If a state $q \in C(i)$, $1 \leq i \leq n$, is in the funnel corresponding to edge e, then q is reachable from the out-node of e via word $f(i)$ (as in (3)), and q is not reachable from the out-node of e using any other word of length $\lceil \log m \rceil$.
(F3) There exist states $p \in B(i)$ and $q \in C(j)$ belonging to the same funnel if and only if $(i, j) \in \omega_n$, $1 \leq i, j \leq n$.

Since $f(i) \neq f(j)$ when $i \neq j$, from (F1) we get the following property for any state $q \in Q$:

If q belongs to a funnel, then q cannot be in sets $B(i)$ and $B(j)$, $i \neq j$. (4)

Recall that states belonging to a funnel associated with edge e can reach e using a path of length $\lceil \log m \rceil$ (or can be reached from e using a path of length $\lceil \log m \rceil$). Analogously (F2) implies the following for any $q \in Q$:

If q belongs to a funnel, then q cannot be in sets $C(i)$ and $C(j)$, $i \neq j$. (5)

By the *count* of a set of funnels $\mathcal{F}$ we mean the cardinality of the set

$$\text{pairs}(\mathcal{F}) = \{(i, j) \mid (\exists p \in B(i))(\exists q \in C(j))(\exists F \in \mathcal{F}) : \\ p \text{ is an in-node of } F \text{ and } q \text{ is an out-node of } F\ \}.$$

Now conditions (4) and (5) together with (F3) give us the following property concerning the set of all funnels of A.

Claim 3.1. *If $\mathcal{F}$ consists of all funnels of the NFA A, then the count of $\mathcal{F}$ equals to the cardinality of ω_n (as a subset of $[n] \times [n]$).*

Now the crucial question is how does the count of a set of funnels relate to the number of transitions that the NFA needs to "realize" the funnels. Note that relations like $\text{less\_than}_n = \{(i,j) \mid 1 \leq i < j \leq n\}$ or $\text{in\_equal}_n = \{(i,j) \mid i \neq j, 1 \leq i,j \leq n\}$ require $\Omega(n^2)$ connections through the funnels and, in light of the seemingly strong properties (F1), (F2), (F3), such languages might look potentially promising for establishing non-linear lower bounds.

It is easy to see that the languages $L(\text{less\_than}_n)$ and $L(\text{in\_equal}_n)$ can be recognized by NFAs having $O(n \cdot \log n)$ transitions. However, it turns out to be difficult to prove lower bounds, and depending on the distribution of letters in the words $f(i)$, the languages could conceivably be recognized using much fewer transitions. In the next subsection we consider binary relations that actually give a much better lower bound than $\Omega(n \cdot \log n)$.

3.2 One-Overlapping Families

Here we define a family of binary relations ω_n, $n \geq 1$, such that $\text{tc}(L(\omega_n))$, $n \geq 1$, is provably not linear in $\text{nsc}(L(\omega_n))$. First we need some more definitions.

For $n > k \geq 1$, a *one-overlap (n,k)-family* is a collection of n subsets of $[n]$,

$$\Delta(n,k) = \{D_1^{(n)}, \ldots, D_n^{(n)}\},\ \ D_i^{(n)} \subseteq [n],\ i = 1, \ldots, n, \tag{6}$$

such that $|D_i^{(n)}| = k$, $i = 1, \ldots, n$, and $|D_i^{(n)} \cap D_j^{(n)}| \leq 1$ for all $i \neq j$, $1 \leq i,j \leq n$.

We define $\phi : \mathbb{N} \longrightarrow \mathbb{N}$ by setting $\phi(n) = k_n$, where k_n is the greatest integer such that a one-overlap (n,k_n)-family exists. It is easy to verify an $\Omega(\log n)$ lower bound for the function $\phi(n)$, however, we can get a better lower bound by relying on results from combinatorics.

We recall that a finite projective plane of order q consists of a set of q^2+q+1 elements called "points" and a set of q^2+q+1 "lines" each consisting of $q+1$ points such that any two lines meet at a unique point, see, e.g., Cameron [2] or Jukna [17].

It is known that when q is a power of a prime number, a projective plane of order q exists [2,17]. Furthermore, when q is a prime power there is an elegant construction for a projective plane of order q. The construction depends on the existence of a finite field $\text{GF}(q)$ containing precisely q elements. For details see, e.g., Anderson [1]. Thus, when q is a prime power,[2]

$$\phi(q^2+q+1) \geq q+1. \tag{7}$$

[2] When q is a prime power, the relation (7) has, in fact, equality by the De Bruijn–Erdös theorem [2].

Corollary 3.1. $\phi(n) \in \Omega(\sqrt{n})$.

In the following consider the case where n is of the form

$$n = q^2 + q + 1, \text{ with } q \text{ a prime power.} \tag{8}$$

Let $\Delta(n, \phi(n)) = \{D_1^{(n)}, \ldots, D_n^{(n)}\}$ be a fixed one-overlap $(n, \phi(n))$-family as in (6). Define

$$\omega_n = \{(i, j) \mid j \in D_i^{(n)}\}. \tag{9}$$

Let $A = (\Sigma, Q, q_0, Q_F, \delta)$ be an arbitrary NFA recognizing the language $L(\omega_n)$, where ω_n is as in (9). Using the set of funnels of A, as defined in the previous subsection, we provide a lower bound for the number of transitions of A.

Since ω_n is defined using the one-overlapping family $\Delta(n, \phi(n))$, the condition (F3) implies that for any funnel F of A one of the following conditions holds:

– there exists $i \in [n]$ such that all in-states of F are in $B(i)$, or
– there exists $j \in [n]$ such that all out-states of F are in $C(j)$.

By the *graph representation of a set of funnels* $\mathcal{F}$ we mean the smallest subgraph G of the state graph of the NFA A such that, for all $F \in \mathcal{F}$, G contains all paths from any in-node of F to any out-node of F.

Claim 3.2. *For any set of funnels $\mathcal{F}$, the number of edges in the graph representation of $\mathcal{F}$ is greater or equal to the count of $\mathcal{F}$.*

Proof. We prove the claim using induction on the number of funnels in $\mathcal{F}$. By properties (4) and (5), the claim clearly holds for any single funnel (actually, an individual funnel needs many more edges, see Remark 3.1 below).

Inductively, assume that the count of a collection of funnels $\mathcal{F} = \{F_1, \ldots, F_k\}$ is not greater than the number of edges in the graph representation of $\mathcal{F}$, and consider a collection of funnels $\mathcal{F}' = \{F_1, \ldots, F_k, F_{k+1}\}$.

Without loss of generality we assume that F_{k+1} has in-states in only one set $B(x)$, $x \in \{1, \ldots, n\}$, and out-states in sets

$$C(j_1), \ldots, C(j_r), \tag{10}$$

where $j_1, \ldots, j_r$, are pairwise distinct, $r \geq 1$. The other possibility is completely symmetric.

We consider the two situations where F_{k+1} can share some out-states with a funnel $F_i \in \mathcal{F}$, $1 \leq i \leq k$.

(i) Consider the case where F_i, $1 \leq i \leq k$, has an in-state in $B(x)$. Now, if F_i has out-states in some set $C(j_s)$ as in (10), $1 \leq s \leq r$, the out-states of F_{k+1} in the set $C(j_s)$ do not increase the count of the set of funnels $\mathcal{F}'$ since the pair (x, j_s) is already included in the count of $\mathcal{F}$. Thus in this case, it does not matter if corresponding to j_s we do not need to include an additional edge for the graph representation of $\mathcal{F}'$.

(ii) Next we consider the case where F_{k+1} shares out-states with some funnel F_i, $1 \le i \le k$, that is not as above in (i), i.e.,

$$F_i \text{ does not have in-states in } B(x). \tag{11}$$

In this case F_{k+1} can share with F_i out-states in only one of the sets $C(j_s)$ as in (10), $1 \le s \le r$. This means that, in order to connect the state(s) of $C(j_s)$ with the state(s) of $B(x)$, the graph representation of $\mathcal{F}'$ needs for each shared set $C(j_s)$ at least one edge that does not appear in the graph representation of $\mathcal{F}$.
Above, note that if the graph representation of $\mathcal{F}'$ would try to "peel off", using a single edge, from the graph representation of $\mathcal{F}$ some state(s) belonging to $C(j_{s_1})$ and some state(s) belonging to $C(j_{s_2})$, $j_{s_1} \neq j_{s_2}$, then these states need to be connected in the graph representation of $\mathcal{F}$. Thus the states would need to be out-states of one funnel $F_i \in \mathcal{F}$, where F_i is as in (11) (since the other possibility was considered in (i) before). Let $y \in [n]$, $y \neq x$, be such that the in-states of F_i are in $B(y)$. Then $j_{s_1}, j_{s_2} \in D_x^{(n)} \cap D_y^{(n)}$, which is a contradiction since $\{D_1^{(n)}, \dots, D_n^{(n)}\}$ is a one-overlap $(n, \phi(n))$-family. The situation in this case is summarized in Figure 1.

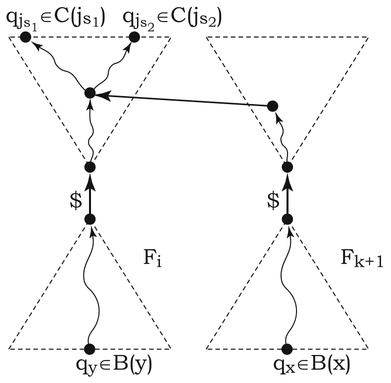


Fig. 1. Case (ii) of the proof of Claim 3.2

Above in (i) and (ii) we have seen that for each pair (x, j_s) that is not already included in the count of $\mathcal{F}$, the graph representation of $\mathcal{F}'$ needs at least one edge that does not occur in the graph representation of $\mathcal{F}$. It follows that the count of $\mathcal{F}'$ is not greater than the number of edges in the graph representation of $\mathcal{F}'$. $\square$

Remark 3.1. In the proof of Claim 3.2, note that an individual funnel F naturally needs essentially more edges than the count of $\{F\}$. In particular, if F has a single in-state (or a single out-state) this state has to be connected to the funnel edge by a path of length $\lceil \log m \rceil$. However, when considering a set of funnels we cannot guarantee that the different paths of length $\lceil \log m \rceil$ would not overlap, and therefore they are not included in the estimate.

Theorem 3.1. *Let ω_n be as in (9), where n is as in (8). Then*

$$\mathrm{tc}(L(\omega_n)) \in \Omega(\mathrm{nsc}(L(\omega_n)) \cdot \sqrt{\mathrm{nsc}(L(\omega_n))}). \tag{12}$$

Proof. Let A_n be any NFA for $L(\omega_n)$ and let $\mathcal{F}$ be the set of all funnels of A_n. By Claim 3.2, $\mathrm{tc}(L(\omega_n))$ is at least the count of $\mathcal{F}$.

Claim 3.1 gives that the count of $\mathcal{F}$ equals the cardinality of ω_n which is $n \cdot \phi(n)$. By Corollary 3.1, $\mathrm{tc}(L(\omega_n)) \in \Omega(n \cdot \sqrt{n})$. The relation (12) holds since by Lemma 3.1 we know that $\mathrm{nsc}(L(\omega_n)) \in O(n)$. □

Except when q is a prime power, very little is known about the existence of projective planes of order q. They do not exist for values 6 or 10. It is probably difficult to find good estimates for $\phi(n)$ except by relying on the existence of projective planes.

By modifying the definition of ω_n when n is not of a form $q^2 + q + 1$ for a prime power q, in Theorem 3.1 we can easily define ω_n for all values of n such that $\mathrm{tc}(L(\omega_n))$ will have a monotonic lower bound with respect to n. Instead of (9) we can define ω_n using any collection of n subsets of $[n]$ where the intersection of any two sets has cardinality at most one. Note that the proof of Theorem 3.1 does not use the property that the subsets have uniform size. In this way we can construct relations ω_n with monotonically increasing cardinality that is guaranteed to reach $n \cdot \sqrt{n}$ infinitely often.

The languages $L(\omega_n)$ are defined over a four-letter alphabet. It is easy to see that encoding the symbols over a binary alphabet does not cause any problems with the above argument that establishes the lower bound. It is sufficient to take in place of α an infinite binary sequence in which any prefix of length n has only one occurrence of any subword of length $c \cdot \lceil \log n \rceil$, for some constant c, and encode the separation markers \$ by a binary sequence that does not occur anywhere else in the words of the language.

Corollary 3.2. *We can construct languages L_n, $n \geq 1$, over a binary alphabet such that* $\mathrm{nsc}(L_n) \in O(n)$ *and* $\mathrm{tc}(L_n) \in \Omega(n \cdot \sqrt{n})$.

To conclude this section, we compare the lower bound with the earlier result due to Hromkovič and Schnitger [14]. By a ε-NFA we mean an NFA that can have ε-transitions. The following constructive lower bound is established in [14].

Proposition 3.1. [14] *There exist regular languages L_n, $n \geq 1$ (with an explicit definition provided), over a binary alphabet recognized by ε-NFAs having $O(n \cdot \log n)$ transitions such that any NFA without ε-transitions needs at least $n \cdot 2^{c \cdot \sqrt{\log n}}$ transitions for every $c < 1/2$.*

When viewing transition complexity as a function of nondeterministic state complexity, the lower bound from Corollary 3.2 is better. The result of Proposition 3.1 was developed for a different purpose, namely to measure the overhead involved in eliminating ε-transitions from NFAs.

4 Strict Transition Complexity

Here we study trade-offs between the number of states and the number of transitions, i.e., the situations where $\text{stc}_k(L)$, $k \geq 0$, may be much larger than $\text{tc}(L)$.

We show that, for any fixed $k \geq 1$, there exist languages L_n, $n \geq 2$, having $(k-1)$-strict transition complexity in $\Omega(n^2)$ but by allowing the use of one more state, i.e., a total of k "additional" states compared to the size of the state minimal NFA, the number of transitions can be reduced to $O(n)$.

Earlier Gruber and Holzer [7] have given examples of languages L'_n where $\text{nsc}(L'_n) = 3n$ and the state minimal NFA for L'_n needs $\Omega(n^2)$ transitions, whereas L'_n has a DFA with $4n+1$ states and $6n+1$ transitions.

Let $n \geq 2$. Denote $L_{1,n} = (a^{n-1}b)^*\text{pref}(a^{n-1}b) - (a^{n-1}b)^*$ and $L_{2,n} = \text{suf}(c^{n-1}d)(c^{n-1}d)^*\text{pref}(c^{n-1}d) \cap \{c,d\}^+$. Let $h : \{c,d\}^* \to \{a,b\}^*$ be the morphism defined by $h(c) = a$, $h(d) = b$. Note that $h(L_{2,n}) \neq L_{1,n}$ since in $h(L_{2,n})$ the first occurrence of b may be preceded by fewer than $n-1$ symbols a. Now we define for $k \geq 1$,

$$L_n[k] = \begin{cases} L_{1,n}(L_{2,n} \cdot h(L_{2,n}))^{(k-1)/2} L_{2,n} & \text{when } k \text{ is odd,} \\ L_{1,n}(L_{2,n} \cdot h(L_{2,n}))^{k/2} & \text{when } k \text{ is even.} \end{cases} \tag{13}$$

Intuitively, $L_n[k]$ is obtained by catenating to the language $L_{1,n} \cdot L_{2,n}$, $k-1$ marked copies of $L_{2,n}$ where consecutive copies are alternately over alphabets $\{a,b\}$ and $\{c,d\}$.

Lemma 4.1. *Let $k \geq 1$ be fixed. For languages $L_n[k]$, $n \geq 2$, as in (13) we have*

(i) $\text{nsc}(L_n[k]) = (k+1)n$,
(ii) $\text{stc}_k(L_n[k]) \in O(n)$,
(iii) $\text{stc}_{k-1}(L_n[k]) \in \Omega(n^2)$.

Due to length restrictions the technical proof of Lemma 4.1 is omitted (we refer the reader to [3] for the proof). By Lemma 4.1 we have the following result.

Theorem 4.1. *Let $k \geq 1$ be fixed. There exist regular languages $L_n[k]$, $n \geq 2$, such that*

(i) $\text{stc}_k(L_n[k]) \in O(\text{nsc}(L_n[k]))$,
(ii) $\text{stc}_{k-1}(L_n[k]) \in \Omega((\text{nsc}(L_n[k])^2)$.

The languages $L_n[k]$ in (13) are defined over a four letter alphabet. Using any reasonable encoding, the result of Theorem 4.1 holds also for languages over a binary alphabet.

Note that Theorem 4.1 (i) and (1) imply that also $\text{tc}(L_n[k]) \in O(\text{nsc}(L_n[k]))$. In particular, by Lemma 4.1, for any fixed integer k there exist languages L_n, $n \geq 2$, with $\text{tc}(L_n) \in O(n)$, such that any NFAs for L_n with at most $\text{nsc}(L_n) + k$ states need $\Omega(n^2)$ transitions.

Finally, we show that there exist families of regular languages where the number of "additional" states in transition-minimal NFAs is not bounded by any constant. We show that the number of states required by transition minimal NFAs for a family of regular languages may be c times the size of the state minimal NFAs for the same languages, for some $c > 1$.

Theorem 4.2. *There exist regular languages L_n, $n \geq 2$, such that* $\mathrm{nsc}(L_n) = 5n - 3$ *and any transition minimal NFA for L_n has at least* $6n - 4$ *states.*

Proof. Let $\Gamma = \{a, b, c, d, e\}$, $\Sigma = \Gamma \cup \{\$\}$ and define

$$L_n = \{x^i \$ y^j \mid x \in \{a, b, c\},\ y \in \{d, e\},\ i + j = n\} \cup \{x^n \mid x \in \Gamma\}.$$

Consider the set of pairs of words $P_n = \{(\varepsilon, a^n), (a^n, \varepsilon)\} \cup \{(x^i, x^j) \mid x \in \Gamma,\ i+j = n,\ 1 \leq i, j \leq n-1\}$. The set P_n satisfies the conditions of the fooling set lower bound technique of, e.g., Hromkovič [12] or Glaister and Shallit [4, Thm. 1]. It follows that any NFA for L_n needs at least $|P_n| = 5n - 3$ states. On the other hand, it is easy to construct an NFA for L_n ($n \geq 2$) with $5n - 3$ states and therefore $\mathrm{nsc}(L_n) = 5n - 3$.

Let $A = (\Sigma, Q, q_0, Q_F, \delta)$ be any NFA for L_n. We say that a state $q \in Q$ has *depth* r, $r \geq 0$, if q is reachable from q_0 on input w where w has exactly r symbols of Γ (that is, the symbol \$ is ignored when considering depth). By the *depth* of a transition $(q_1, x, q_2) \in \delta$, $q_1, q_2 \in Q$, $x \in \Sigma$, we mean the depth of q_1. Since all words of L_n have exactly n symbols of Γ (and all states of Q are useful), it follows that any state of Q and any transition of δ has a unique depth.

In the following we outline the argument used in the proof (for details the reader is referred to the full version available in [3]). It can be verified that, for each $1 \leq i \leq n - 1$, A has at least 5 states of depth i. If A has only 5 states of depth i, A needs 11 transitions of depth i but by introducing one additional state of depth i the number of depth i transitions can be reduced to 10.

Since we have observed that each state has a unique depth, the changes made for depth i states and transitions are localized to depth i. Thus, an NFA that uses a minimal number of transitions has to have at least 6 states of each depth $1 \leq i \leq n - 1$. □

5 Conclusion

We have given an explicit construction of regular languages L_n, $n \geq 1$, having nondeterministic state complexity in $O(n)$, such that the transition complexity of L_n is $\Omega(n^{\frac{3}{2}})$. This does still not reach the lower bound $\Omega(\frac{n^2}{\log n})$ obtained using probabilistic combinatorial methods [9,18]. Naturally the main open question is whether it is possible to prove that for all families of regular languages L_n, $n \geq 1$, with $\mathrm{nsc}(L_n) \in O(n)$, the transition complexity satisfies $\mathrm{tc}(L_n) \in o(n^2)$.

We would like to have more general purpose tools for proving transition complexity lower bounds, in the spirit of the techniques considered in [4,12] for nondeterministic state complexity lower bounds. It remains to be seen whether the notions associated with funnels and one-overlapping families that were introduced for the result of Theorem 3.1 can be extended in this way.

Theorem 4.2 gives a family of regular languages L_n where any transition minimal NFA for L_n has at least $c \cdot \mathrm{nsc}(L_n)$ states where $c \approx 6/5$. By increasing the size of the alphabet, the constant c can be somewhat increased. However, we do not know how to construct a language L, where the transition minimal NFA would need, for example, $2 \cdot \mathrm{nsc}(L)$ states.

References

1. Anderson, I.: Combinatorial Designs, Construction Methods. John Wiley (1990)
2. Cameron, P.J.: Combinatorics: Topics, Techniques, Algorithms. Cambridge University Press (1996)
3. Domaratzki, M., Salomaa, K.: Lower bounds for the transition complexity of NFAs. Queen's School of Computing Technical Report No. 2006-515. Available at `www.cs.queensu.ca/TechReports`
4. Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inform. Proc. Lett. **59** (1996) 75–77
5. Goldstine, J., Kappes, M., Kintala, C.M.R, Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. Universal Comput. Sci. **8** (2002) 193–234
6. Gramlich, G., Schnitger, G.: Minimizing NFA's and regular expressions. In: Proc. STACS 2005. Lect. Notes Comput. Sci., Vol. 3404. Springer–Verlag (2005) 399–411
7. Gruber, H., Holzer, M.: A note on the number of transitions of nondeterministic finite automata. In: Fernau, H.: (ed.): 15. Theorietag der GI-Fachgruppe 0.1.5 "Automaten und Formale Sprachen" (2005) 24–25
8. Gruber, H., Holzer, M.: Finding lower bounds for nondeterministic state complexity is hard. In: Ibarra, O.H., Dang, Z. (eds.): Proceedings DLT 2006. Lect. Notes Comput. Sci., Vol. 4036. Springer–Verlag (2006)
9. Gruber, H., Holzer, M.: Results on the average state complexity of finite automata accepting finite languages. In: Proc. of Descriptional Complexity of Formal Systems, DCFS 2006, to appear
10. Holzer, M., Kutrib, M.: Unary language operations and their nondeterministic state complexity. In: Proc. DLT 2002. Lect. Notes Comput. Sci., Vol. 2450. Springer–Verlag (2003) 162–172
11. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Internat. J. Foundations of Computer Science **14** (2003) 1087–1102
12. Hromkovič, J.: Communication Complexity and Parallel Computing. Springer–Verlag (1997)
13. Hromkovič, J.: Descriptional complexity of finite automata: Concepts and open problems. J. Automata, Languages and Combinatorics **7** (2002) 519–531
14. Hromkovič, J., Schnitger, G.: NFAs with and without ε-transitions. Proc. ICALP 2005. Lect. Notes Comput. Sci., Vol. 3580. Springer–Verlag (2005) 385–396
15. Hromkovič, J., Seibert, S., Wilke, T.: Translating regular expressions in small ε-free nondeterministic finite automata. J. Comput. System Sci. **62** (2001) 565–588
16. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. **22** (1993) 1117–1141
17. Jukna, S.: Extremal Combinatorics with Applications in Computer Science. EATCS Texts in Theoretical Computer Science, Springer–Verlag (2001)
18. Kari, J.: Personal communication, April 2006
19. Lifshits, Yu.: A lower bound on the size of ε-free NFA corresponding to a regular expression. Inform. Proc. Lett. **85** (2003) 293–299
20. Malcher, A.: Minimizing finite automata is computationally hard. Theoret. Comput. Sci. **327** (2004) 375–390
21. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds): Handbook of Formal Languages, Vol. I. Springer–Verlag (1997) 41–110
22. Yu, S.: State complexity of finite and infinite regular languages. Bulletin of the EATCS **76** (2002) 142–152

Smart Robot Teams Exploring Sparse Trees*

M. Dynia[1], J. Kutyłowski[2], F. Meyer auf der Heide[3], and C. Schindelhauer[4]

[1] DFG Graduate College "Automatic Configuration in Open Systems",
Heinz Nixdorf Institute, University of Paderborn, Germany
`mdynia@uni-paderborn.de`

[2] International Graduate School of Dynamic Intelligent Systems,
Heinz Nixdorf Institute, University of Paderborn, Germany
`jarekk@uni-paderborn.de`

[3] Heinz Nixdorf Institute, University of Paderborn, Germany
`fmadh@uni-paderborn.de`

[4] Computer Networks and Telematics, University of Freiburg, Germany
`schindel@informatik.uni-freiburg.de`

Abstract. We consider a tree which has to be completely explored by a group of k robots, initially placed at the root. The robots are mobile and can communicate using radio devices, but the communication range is bounded. They decide based on local, partial knowledge, and exchange information gathered during the exploration. There is no central authority which knows the graph and could control the movements of the robots – they have to organize themselves and jointly explore the tree.

The problem is that at every point of time the remaining unknown part of the tree may appear to be the worst case setting for the current deployment of robots. We present a deterministic distributed algorithm to explore T and we use a parameter of a tree called *density*. We compare the performance of our algorithm with the optimal algorithm having a-priori knowledge of the same tree. We show that the above ratio is influenced only by the density and the height of the tree. Since the competitive ratio does not depend on the number of robots, our algorithm truly emphasizes the phenomena of self-organization. The more robots are provided, the faster the exploration of the terrain is completed.

1 Introduction

We study the problem of exploration of trees by a group of k mobile robots equipped with radio communication devices. The robots can move and can also communicate using radio devices, but the communication range is bounded. This allows a robot only a local view of the situation. Hence it has to make all of its decisions basing only on partial knowledge. One robot cannot explore the graph fast, so the group exchanges information gathered during the exploration. There

* This research is partially supported by the DFG-Sonderforschungsbereich SPP 1183: "Organic Computing. Smart Teams: Local, Distributed Strategies for Self-Organizing Robotic Exploration Teams" and by the EU within the 6th Framework Programme under contract 001907 (DELIS).

is no central authority which knows the graph and could control the robots, so the team has to organize itself and jointly explore the tree, i.e. visit all its nodes. The problem is that at every point of time the remaining unknown part of the tree may appear to be the worst case setting for the current deployment of robots.

We use competitive analysis [1] to measure the performance of the algorithm. Let $C_{\mathcal{A}}(T)$ denote the running time of the online distributed algorithm $\mathcal{A}$ exploring a tree T not known in advance. Only outgoing edges are visible to a robot placed in some node, and it has to traverse an edge in order to recognize new parts of the graph. Since at each point of time the graph is known only partially, the adversary may construct the worst possible remaining part of the graph in an online fashion. In this model, traversing an edge takes one time step and costs of all other operations are neglected.

Let $C_{\mathrm{OPT}}(T)$ be the running time of the optimal algorithm which knows the exact structure of T and explores it optimally. The algorithm $\mathcal{A}$ is σ-*competitive* if for all T

$$C_{\mathcal{A}}(T) \le \sigma \cdot C_{\mathrm{OPT}}(T) + \alpha \tag{1}$$

for some constant α.

Since robots use wireless devices with a bounded communication radius $\mathcal{A}$ is allowed to use only local communication, i.e. robot can communicate (once per time step) only with those robots which are at distance one or less in the tree.

We can observe that the optimal offline algorithm does not have to use communication at all, since the graph is given in advance.

1.1 Related Work

The problem of exploring an unknown environment has been widely studied (see [2] for a survey) and usually the environment is modeled as a graph. Famous Traveling Salesman Problem and Chinese Postman Problem expose main problems of graph exploration.

The k-Traveling Salesman Problem (k-TSP) and k-Chinese Postman Problem (k-CPP) [3] are interesting extensions of these well known NP-hard problems[1]. All nodes (and edges) of a graph have to be visited (at least once) by one of the k robots initially placed at some node of the graph, subject to minimize the maximum length route of a robot. The contribution [5] captures the hardness of the k-TSP and shows how to construct (in polynomial time in size of a graph but exponential in k) optimal routes for an arbitrary k. In the location-allocation version of this problem initial positions of robots have to be found as well.

There are several approximation algorithms of minmax k-TSP on a tree presented in [6,4,3,7,8,5], but we should mention here that finding k-approximation is easy, and yields by taking a trivial optimal solution of 1-TSP (DFS traversal of a tree). Frederickson, Hecht and Kim [3] gave first worst-case analysis of k-TSP and developed $2 - 1/k$ approximation algorithm.

[1] Also often called p-TSP and p-CPP, e.g. in [4].

In this work we consider an *online* version of k-TSP where a labeled graph is not known in advance, and the goal is to minimize overall time of an exploration. Authors of [9] and [10] study even harder problem and assume an unlabeled graph. If this setting one robot cannot explore the graph alone. Hence, in [10] the robot puts *pebbles* on the nodes of the graph in order to recognize visited ones, and in [9] it cooperates with other robot by exploration.

There are many publications considering an online exploration by a *single* robot (e.g. [11,12,13,14]), but the problem which exposes a real flavor is considering an exploring group of k robots [15,16,5].

Profits from a collective online exploration of *trees* are investigated in [15]. The authors prove competitive ratio of $\mathcal{O}(k/\log k)$ for the time of exploration compared to the time of an optimal algorithm which knows the tree. They also prove a lower bound of $2 - 1/k$ for this ratio. Moreover they study the influence of communication on the complexity of collective exploration. They prove that when no communication allowed, the competitive ratio is at least $\Omega(k)$. This shows that there is no cooperation at all, if no communication granted – the team explores in the same time that one robot would need to do it trivially.

Authors of [5] develop from their 2-approximation algorithm (similar to this in [3]) an *online* algorithm, subject to minimize *energy* used by a robot, rather than *time* of an exploration. They prove it uses at most 8 times the energy per robot the optimal strategy for k robots does. They prove a lower bound of 1.5 for this energy model, which differs from the time model investigated in our work and in [15].

1.2 Our Results

We investigate a model with the cost to be an overall time of the collective tree exploration. Our results are two distributed, local algorithms, which explore an arbitrary tree using a group of k robots, and both exhibit a competitive ratio independent of k. It will solely depend on the tree, especially on its height and so called *density.*

Let us first define the *density* $p(T) \in \mathbb{N}$ for some tree T. It is the minimal natural number with the following property

$$\forall_{T'=(V',E')\subset T} \quad |V'| \leq 4 \cdot [h(T')]^{p(T)} \tag{2}$$

where $h(T)$ is the height of T. For many classes of trees an upper bound for $p(T)$ is known. For example trees embedded into a planar grid, fulfill $p(T) \leq 2$, and for binary trees $p(T) \leq h(T)/\log h(T)$. The density influences the running time of our algorithm. For simplicity of notation we will use D to denote the height $h(T)$ and p to denote $p(T)$.

The first distributed algorithm is presented in Sect. 2.2. It consists of two sub-algorithms, described and analyzed in the same section. It assumes only local communication and is fully online, i.e. no previous knowledge of the tree is required. It achieves a competitive ratio of

$$\mathcal{O}\left(D^{1-1/p} \cdot \min\left\{p, \log p \cdot D^{1/2p}\right\}\right)$$

where p is the density of the tree and D is its height.

The second algorithm described in Sect. 2.3 is also distributed and local. It has an improved competitive ratio of

$$\mathcal{O}\left(D^{1-1/p}\right) ,$$

but unfortunately it requires knowledge of the density value (or at least its constant approximation) before the exploration. It may be considered to be a drawback, but in fact, the value of density is known for many classes of trees.

For an arbitrary tree T we have $p(T) \leq \log n$ and thus our first algorithm is $\mathcal{O}(D \log \log n)$-competitive and the second algorithm is $\mathcal{O}(D)$-competitive. When we consider a class of trees which can be embedded in the s-dimensional grid it is easy to observe an upper bound $p(T) \leq 2s$. However, according to the definition (2) we have $p(T) \leq 2$ for a 2-dimensional grid and thus our algorithm is $\sqrt{D}$ competitive for this class of trees.

Unlike in [15], we present an algorithm with a competitive ratio which does not grow for large number of robots. It results in improved overall time of exploration for a team with increased number of robots. The coordination problem does not appear and thus our algorithm truly emphasizes the phenomena of robots cooperation. Additionally, our algorithms have an advantage over the one in [15], namely they are fully distributed and use only local communication. We prove that the exploration can also be efficient even under very bounded communication scheme.

2 The Online Algorithm

Only edges outgoing from the root are visible to a robot initially placed in the root of T, and it has to traverse an edge in order to recognize new parts of the graph. In our model, traversing an edge takes one time step and we neglect costs of all other operations. Robots communicate only if they meet in the same node – in this case in the root.

First, we present in Sect. 2.2 the *KarlsruheExpress-Clever* algorithm (*KE-Clever*) and then in Sect. 2.3 the *KE-Oblivious* algorithm. Both algorithms took their name from the place where the first steps of their design were made.

All presented algorithms are based on the same idea. Robots start in the root and explore the tree in so called chunks. They leave the root, enter the tree and after a current chunk is explored, they return to the root in order to exchange a valuable information and agree upon the strategy of exploration of the next chunk. Consecutive chunk lays further from the root than its predecessor – and in this way the progress of the exploration is realized. At some point of time the furthest leaf is reached by a robot and thus the tree is completely explored.

The team needs the information on the density of the tree in order to accurately estimate the amount of work a group can handle, i.e. the size of the chunk. The higher the density of the graph, the more job to do for robots, and thus the size of a chunk has to be reduced. On the other hand, small chunks weaken the progress of the exploration. The only aspect which distinguishes one algorithm from the other is the way they look for a good estimation of the destiny value. A proper balance between the size of the chunk and the efficiency of the exploration has to be found.

2.1 Definitions

The description of the algorithms as well as their performance analysis uses some notion which we define in this section (the circumference function ϕ, the *MyDFS* routine and finally the function r_{new}).

Given a rooted tree $T = (V, E)$ we denote by $h(T)$ the height of the tree, i.e. the number of nodes on a longest path from the root to a leaf. The *level* r of the tree is a set of nodes in distance r from the root. We describe nodes on each level, such that $v_r(i)$ denote the i-th node on level $r \in \mathbb{N}$ of T and $\phi(r)$ the number of nodes on that level. Let $T_v(h)$ be the subtree of T rooted at v and h in height, and let it be the subtree with maximal number of edges. Then, for some node $v \in V$ we define by *MyDFS*(v, r, p) the sequence of nodes created by the concatenation of the following sequences of nodes (Fig. 1) as

- the path from the root of T to the node v,
- the classical DFS algorithm for the tree $T_v(r^{1/p})$, broken after $8r$ steps (at node v'),
- the path from v' to v, and
- the path from v to the root of T.

We use *MyDFS*(v, r, p) only when the route from the root to the node v of length l is already known. In this case the execution takes $\mathcal{O}(r + l)$ time steps.

Moreover, we use the function r_{new} which dictates the progress of the presented algorithm. It is defined for a tree T and any number $r \geq 1$ and $p \geq 1$

$$r_{\text{new}}(r, p) = \operatorname{argmin}\left\{\phi(j) : \quad j \in \left[r + \frac{1}{2} \cdot r^{1/p},\ r + r^{1/p}\right]\right\} . \tag{3}$$

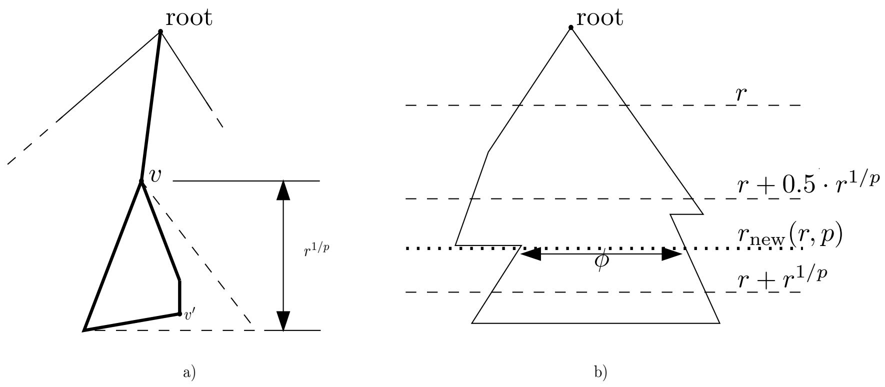


Fig. 1. Definition of a) MyDFS routine and b) the function r_{new}

Intuitively, the value of r_{new} points out the most narrow place in the tree within some specified interval (Fig. 1).

2.2 The *KE-Clever* Algorithm

First we present and analyze in detail the *KE-2* algorithm and then the *KE-1* algorithm which is a simple modification of the former. Both algorithms are the basic components of *KE-Clever* which we define and analyze at the end of this section.

The *KE-2* algorithm works in so called *epochs.* During one epoch all nodes in distance between some two well defined levels (*a chunk*) are being explored. A detailed description of the *KE-2* provides Algorithm 1. and Figure 2.

Algorithm 1. The *KE-2* algorithm

```
Require: $k$ robots placed in $ROOT$
 1: $r \leftarrow 1$
 2: $p' \leftarrow 1$
 3: repeat
 4:   $\phi \leftarrow \phi(r)$
 5:   for ($j \leftarrow 0$ to $\lceil \phi/k \rceil$) do
 6:     $v \leftarrow v_r(ID + k \cdot j \mod \phi)$
 7:     repeat
 8:       if (some $RISE$ flag is set) then
 9:         $p' \leftarrow 2 \cdot p'$
10:       end if
11:       move and explore following $MyDFS(v, r, p')$ sequence
12:       if ($T_v(r^{1/p'})$ is not completely explored) then
13:         set $RISE$ flag
14:       end if
15:     until (no flag $RISE$ is set)
16:   end for
17:   $r \leftarrow r_{\text{new}}(r, p')$
18: until ($T$ is completely explored)
```

The algorithm maintains a variable r (the radius) which value means that all levels of the tree up to the level r are already explored. It grows during the exploration, and the algorithm terminates when the tree is completely known. We call the lines 4–17 of the code an *epoch* of the algorithm. Epoch E_i starts with radius r_i and during the epoch all nodes on levels between r_i and r_{i+1} (*a chunk* C_i) are being explored.

The following lemma demonstrates an idea of the algorithm and shows that the exploration does progress and additionally shows that the number of epochs is small.

Lemma 1. *The* KE-2 *terminates after executing* $\mathcal{O}(D^{1-1/2p})$ *epochs, where* $p = p(T)$ *is the density of* T.

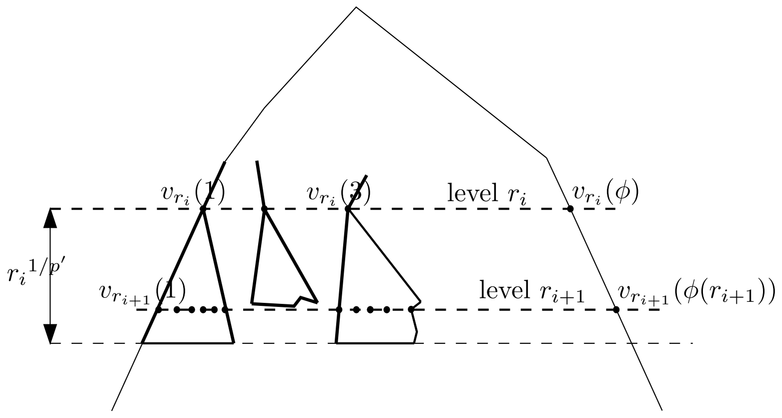


Fig. 2. One epoch of the *KE-2* algorithm

Proof. We start by showing a property of some sequence a_i which suitably describes the behavior of our algorithm. Then using this property we show the upper bound on the number of epochs. We show that for $a_0 = 0, a_1 = 1$ and $a_{i+1} = a_i + \frac{1}{2} \cdot {a_i}^{1/2p}$ we have

$$a_i \geq \left(\frac{i}{6}\right)^{2p/(2p-1)} . \tag{4}$$

Indeed, by induction we have the lower bound for $a_{i+1} = a_i + \frac{1}{2} \cdot a_i^{1/2p}$ of

$$(i/6)^{2p/(2p-1)} + \frac{1}{2} \cdot \frac{i^{1/(2p-1)}}{6^{1/(2p-1)}} \geq (i/6)^{2p/(2p-1)} + \frac{3 \cdot i^{1/(2p-1)}}{6^{2p/(2p-1)}} \geq \frac{(i+3)i^{1/(2p-1)}}{6^{2p/(2p-1)}} \geq$$

$$\geq \frac{(i+1)(i+1)^{1/(2p-1)}}{6^{2p/(2p-1)}} \geq \left(\frac{i+1}{6}\right)^{2p/(2p-1)} .$$

The main point of the above reasoning is that for $0 < \alpha \leq 1$, $i \geq 0$

$$\frac{i+3}{i+1} \geq \left(\frac{i+1}{i}\right)^{\alpha} .$$

Now let us observe that $r_i \geq a_i$. Therefore, by finding j s.t. $a_j \geq D$ we can argue that j epochs suffice to explore the whole tree. This is the case for $j = (D \cdot 6)^{1-1/2p}$ (we use (4) to show that) and thus at most $\mathcal{O}(D^{1-1/2p})$ epochs are executed. □

The value of the local variable p' at the beginning of the epoch E_i is the same for all robots and defines the value of p_i. By the sequence p_i our distributed algorithm estimates the upper bound for the density of the tree it explores. The lines 6–15 are *a turn* of the algorithm. Each epoch E_i consists of many turns during which subtrees in the chunk C_i are being explored.

All lines of the code (except one: the traversing according to *MyDFS*) are executed by all robots placed in the root of the tree. Then the local communication needed to compute e.g. ϕ and a new value of p' is granted. This emphasizes the locality and distributed property of our algorithm.

We now show that an execution of an epoch takes a small number of parallel steps for the *KE-2* algorithm.

Lemma 2. *The epoch E_i $(i \geq 1)$ of* KE-2 *terminates after*

$$\mathcal{O}\left(\frac{\phi(r_i)}{k} \cdot r_i \cdot [1 + \log(p_{i+1}/p_i)]\right)$$

time steps, after completely exploring the chunk C_i.

Proof. The epoch starts based on level r_i of the tree T, where exactly $\phi(r_i)$ subtrees $T_1(h), \ldots, T_{\phi(r_i)}(h)$ are rooted, where

$$T_j(h) = T_{v_{r_i}(j)}(h) \ .$$

Within one epoch, turn by turn, these subtrees are explored by our algorithm. During this execution the value of h changes. Initially $h = r_i^{1/p_i}$ and it decreases until it reaches $r_i^{1/p_{i+1}}$ at the end of the epoch.

The group of k robots starts the exploration from subtrees rooted at nodes with the smallest IDs. Since p_i is doubled during the exploration, the *repeat-until* loop, inside of the turn, terminates after $1 + \log(p_{i+1}/p_i)$ passes. Indeed, since $p_{i+1} \leq p$ the algorithm will reach some value, i.e. p_{i+1}, which is sufficient to explore all $T_v(r^{1/p_{i+1}})$ by *MyDFS* and then no *RISE* flag is set.
For each j and $p_i \leq p' \leq p_{i+1}$ the *MyDFS*$(v_{r_i}(j), r_i, p')$ algorithm takes at most $11r_i$ steps. This implies that one turn takes $\mathcal{O}(r_i)$ time steps. Moreover, there are exactly $\lceil \phi(r_i)/k \rceil$ turns during an epoch.

This proves that during the epoch E_i the algorithm explores the chunk C_i in time $\mathcal{O}\left(\frac{\phi(r_i)}{k} \cdot r_i \cdot [1 + \log(p_{i+1}/p_i)]\right)$ and eventually places all robots in the root. □

The following lemma describes the running time of the *KE-2* algorithm.

Lemma 3. *The* KE-2 *explores the tree T in time*

$$\mathcal{O}\left(\frac{\log p}{k} \sum_{i \geq 0} \phi(r_i) r_i\right) \ .$$

Proof. By Lemma 2, the epoch E_i needs at most $\phi(r_i)/k \cdot r_i \cdot [1 + \log(p_{i+1}/p_i)]$ time steps. Using $\frac{p_{i+1}}{p_i} \leq 2p$ and summing up over all epochs we have

$$\sum_{i \geq 0} \frac{1}{k} \phi(r_i) \cdot r_i \cdot [1 + \log(p_{i+1}/p_i)] \leq \frac{1}{k} \sum_{i \geq 0} \phi(r_i) \cdot r_i \cdot 2 \log p \ .$$

□

Now we compare the time of our online algorithm to the time the optimal offline algorithm would need for the same tree T. To show this we need to know the running time of the optimal algorithm.

In Lemma 4 we show a lower bound for running time of all algorithms exploring T. We describe this bound by using the sequence r_i and p_i, both defined by an execution of *KE-2*.

Lemma 4. *The optimal algorithm needs*

$$\Omega\left(\frac{1}{k}\sum_{i\geq 0}\phi(r_i){r_i}^{1/p_i}\right)$$

time steps.

Proof. Given the sequence r_i of levels of tree and the sequence p_i defined by the *KE-2*, define the set I_i of levels of T as follows

$$I_i = \left[r_i + \frac{1}{2}\cdot {r_i}^{1/p_{i+1}},\ r_i + {r_i}^{1/p_{i+1}}\right]$$

and let $\mathcal{I} = \bigcup_i I_i$ be the sum of these sets of levels. The levels I_i do not overlap since

$$r_{i+1} + \frac{1}{2}r_{i+1}^{1/p_{i+2}} > r_i + r_i^{1/p_{i+1}}$$

and thus the overall number of nodes at levels described by $\mathcal{I}$ is a lower bound on number of nodes in the tree T.

Let us count how many nodes n_i there are on levels contained in I_i. We know that $r_{i+1} \in I_i$ and that $\forall_{j\in I_i}(\phi(r_{i+1}) \leq \phi(j))$. Then we have $n_i \geq 1/2\cdot\phi(r_{i+1})\cdot {r_i}^{1/p_{i+1}}$ and given $r_i \geq r_{i+1}/2$ we get

$$n_i = \Omega\left(\phi(r_{i+1})\cdot {r_{i+1}}^{1/p_{i+1}}\right)$$

so there are at least $\Omega\left(\sum_i \phi(r_i){r_i}^{1/p_i}\right)$ nodes in the tree.

A group of k robots needs at least s/k time steps to explore a tree with s nodes, which proves that even the optimal algorithm needs the time claimed in the lemma. □

We can prove now the competitive ratio of *KE-2* in the following lemma.

Lemma 5. *The* KE-2 *achieves competitive ratio of* $\mathcal{O}(\log p\cdot D^{1/2p}\cdot D^{1-1/p})$

Proof. First, we show that for any non-decreasing sequence $\{y_i\}$ and any sequence $\{x_i\}$ and for any $0<\alpha<1$ we have

$$\frac{\sum_{i=1}^m x_i y_i}{\sum_{j=1}^m x_i {y_i}^{\alpha}} \leq {y_m}^{1-\alpha}\ . \tag{5}$$

To prove that, we notice

$$\frac{1}{y_m^{1-\alpha}} \cdot \frac{\sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i \cdot y_i^\alpha} \leq 1 \ .$$

Indeed, for all $0 \leq i \leq m$ we have $y_i^{1-\alpha} \leq y_m^{1-\alpha}$ and we can upper-bound this term by

$$\frac{\sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i \cdot y_i^\alpha y_m^{1-\alpha}} \leq \frac{\sum_{i=1}^m x_i y_i}{\sum_{i=1}^m x_i y_i^\alpha y_i^{1-\alpha}} \leq 1 \ .$$

In the remaining part of the proof we lay $\alpha = 1/2p$, $y_i = r_i$ and $x_i = \phi(r_i)$ in (5).

We notice that ${r_i}^{1/p_i} \leq D^{1/2p}$ and combine the results of Lemma 3 and 4 to provide the bound for competitive ratio σ

$$\sigma \leq \mathcal{O}\left(\log(4p) \cdot \frac{\sum_i \phi(r_i) r_i}{\sum_i \phi(r_i) {r_i}^{1/p_i}}\right) \leq \mathcal{O}\left(\log p \cdot D^{1-1/2p}\right) . \qquad \square$$

Now we present an algorithm which is a small modification of *KE-2*. Unlike the former, it will search for the value of density more accurately (not binary) and thus it will be a bit slower. The *KE-1* algorithm arises by replacing in Algorithm 1. the 9-th line of code $p' \leftarrow 2 \cdot p'$ by the new line $p' \leftarrow p' + 1$.

The proof of the performance of *KE-1* is quite the same as the proof of *KE-2* and thus we omit the detailed proof here. The *KE-1* is a distributed and online algorithm and achieves the following competitive ratio

Corollary 1. *The* KE-1 *is* $\mathcal{O}(p \cdot D^{1-1/p})$*-competitive.*

Proof. For the *KE-1* the local variable p' is increased only by one and thus epoch E_i takes

$$\mathcal{O}\left(\frac{\phi(r_i)}{k} \cdot r_i \cdot [1 + p_{i+1} - p_i]\right)$$

time steps. Summing up over all epochs we have $\mathcal{O}\left(p/k \cdot \sum_i \phi(r_i) r_i\right)$ time steps until tree is completely explored by *KE-1*. Since p_i approximates the value of $p(T)$ more accurately, we can use the lower bound for the time of optimal algorithm by Lemma 4, to obtain the competitive ratio of $\mathcal{O}\left(p \cdot D^{1-1/p}\right)$. $\square$

We have now two algorithms *KE-1* and *KE-2* which performance depends on the ratio between p and D. The first one is better for trees with a small density (comparing to D), and the second is better for a small – comparing to the density – height of the tree.

To take advantages of these algorithms and avoid drawbacks, we define the *KE-Clever* in the following way. The robot with the ID 1 (*a referee*) does not move but only waits in the root and serves as a relay station for the communication. It maintains the status of an exploration. Let now the first half of the remaining group of robots (the subgroup A) executes the *KE-1* and the other half (the subgroup B) the *KE-2* algorithm.

The *KE-Clever* terminates, after one of the group reports a completion of the exploration. The group A needs at most $\mathcal{O}\left(p \cdot D^{1-1/p}\right) \cdot C_{\text{OPT}}$ time and the group B needs $\mathcal{O}\left(\log p \cdot D^{1/2p} \cdot D^{1-1/p}\right) \cdot C_{\text{OPT}}$ time, which leads to the following theorem:

Theorem 1. *The* KE-Clever *achieves the competitive ratio of*

$$\mathcal{O}\left(D^{1-1/p} \cdot \min\left\{p, \log p \cdot D^{1/2p}\right\}\right)$$

for an arbitrary tree T, where D is the height of T and p the density.

2.3 The *KE-Oblivious* Algorithm

The algorithms *KE-1* and *KE-2* estimate the density value in a different way, but unfortunately, in the both cases the time needed for searching the proper value reflects unfavorably in their performance. Let us assume that the density $p(T)$ of the tree is known beforehand. Then, replacing in Algorithm 1. the 2-nd line of code $p' \leftarrow 1$ by the new line $p' \leftarrow p(T)$ we define a new algorithm called *KE-Oblivious.*

Theorem 2. *If the density parameter p for the tree is arbitrary but known before the exploration, then the* KE-Oblivious *has competitive ratio of*

$$\mathcal{O}\left(D^{1-1/p}\right)$$

for an arbitrary tree T, D in height.

Proof. Since the density is known in advance, there is no need for approximation. In fact, the local variable p' never changes during the execution of the algorithm. This results in only $\mathcal{O}\left(\frac{\phi(r_i)}{k} \cdot r_i\right)$ time steps needed for an epoch E_i. Following the same approach as in Lemma 3 and 4 we obtain an improved competitive ratio of $\mathcal{O}\left(D^{1-1/p}\right)$. □

3 Conclusions

We have presented two distributed online algorithms which explore the unknown tree starting from the root. Assuming global communication (or allowing landmarks in nodes) one can combine by a simple trick the *KE-Clever* or *KE-Oblivious* with the algorithm described in [15], obtaining the competitive ratio equal to the *minimum* of ratios of each combined algorithm. It is enough to execute them both in parallel – each algorithm executed by the half of the group.

In the face of the best known lower bound of 2-1/k, the problem concerning an optimal online competitive ratio for trees is still open. And finally, does the bounded communication have an impact on that ratio at all?

References

1. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York, NY, USA (1998)
2. Rao, N., Kareti, S., Shi, W., Iyenagar, S.: Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM-12410 (1993)
3. Frederickson, G., Hecht, M., Kim, C.: Approximation algorithms for some routing problems. SIAM Journal on Computing **7** (1978) 178 – 193
4. Averbakh, I., Berman, O.: Minmax p-traveling salesmen location problems on a tree. Annals of Operations Research **110** (2002) 55 – 68
5. Dynia, M., Korzeniowski, M., Schindelhauer, C.: Power-aware collective tree exploration. In Grass, W., ed.: Proceedings of ARCS'06. LNCS 3894, Springer Verlag (2006) 341 – 351
6. Even, G., Garg, N., Könemann, J., Ravi, R., Sinha, A.: Min-max tree covers of graphs. Operations Research Letters **32** (2004) 309 – 315
7. Averbakh, I., Berman, O.: (p - 1)/(p + 1)-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective. Discrete Applied Mathematics **75** (1997) 201 – 216
8. Averbakh, I., Berman, O.: A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. Discrete Applied Mathematics **68** (1996) 17 – 32
9. Bender, M., Slonim, D.: The power of team exploration: two robots can learn unlabeled directed graphs. In: Proc. FOCS 1994. (1994) 75 – 85
10. Bender, M., Fernández, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: exploring and mapping directed graphs. In: Proc. 30th Symp. Theory of Computing, ACM (1998) 269 – 278
11. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. In: SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (1998) 316 – 322
12. Awerbuch, B., Betke, M., Rivest, R., Singh, M.: Piecemeal graph exploration by a mobile robot. Information and Computation **152** (1999) 155 – 172
13. Dessmark, A., Pelc, A.: Optimal graph exploration without good maps. In: Algorithms - ESA 2002: 10th Annual European Symposium. Volume 2461., Springer (2002) 374 – 386
14. Fleischer, R., Trippen, G.: Exploring an unknown graph efficiently. In: 13th Annual European Symposium. Volume 3669., Springer (2005) 11 – 22
15. Fraigniaud, P., Gasieniec, L., Kowalski, D., Pelc, A.: Collective tree exploration. In: Proc. LATIN 2004. Volume 2976. (2004) 141–151
16. Fraigniaud, P., Ilcinkas, D., Rajsbaum, S., Tixeuil, S.: Space lower bounds for graph exploration via reduced automata. In: Structural Information and Communication Complexity: 12 International Colloquium. Volume 3499. (2005) 140 – 154

k-Sets of Convex Inclusion Chains of Planar Point Sets

Wael El Oraiby and Dominique Schmitt

Laboratoire MIA, Université de Haute-Alsace
4, rue des Frères Lumière, 68093 Mulhouse Cedex, France
{Wael.El-Oraiby, Dominique.Schmitt}@uha.fr

Abstract. Given a set V of n points in the plane, we introduce a new number of k-sets that is an invariant of V: the number of k-sets of a convex inclusion chain of V. A convex inclusion chain of V is an ordering $(v_1, v_2, ..., v_n)$ of the points of V such that no point of the ordering belongs to the convex hull of its predecessors. The k-sets of such a chain are then the distinct k-sets of all the subsets $\{v_1, ..., v_i\}$, for all i in $\{k+1, ..., n\}$. We show that the number of these k-sets depends only on V and not on the chosen convex inclusion chain. Moreover, this number is surprisingly equal to the number of regions of the order-k Voronoi diagram of V. As an application, we give an efficient on-line algorithm to compute the k-sets of the vertices of a simple polygonal line, no vertex of which belonging to the convex hull of its predecessors on the line.

1 Introduction

Given a finite set V of n points in the Euclidean plane (no three of them being colinear) and an integer k $(0 < k < n)$, the k-sets of V are the subsets of k points of V that can be strictly separated from the rest by a straight line. Due to the various applications of k-sets, the problems of constructing and of counting them have been extensively studied in computational and combinatorial geometry. Dey [5] has shown that the number $\gamma^k(V)$ of k-sets of a set V of n points in the plane is at most $O(nk^{\frac{1}{3}})$ and Tóth [12] has shown how to construct point sets with $n2^{\Omega(\sqrt{\log k})}$ k-sets. Narrowing the gap between these two bounds remains an important open problem. More precise results have been obtained by adding up the number of k-sets for different values of k. Peck [11] has shown that the number of $(\leq k)$-sets of V, i.e. the sum of the numbers $\gamma^i(V)$ over all i in $\{1, ..., k\}$, is bounded by kn and that this bound is tight. In this paper we propose a different approach which consists in fixing k and summing the number of k-sets over different subsets of V. To this aim, we define the notion of convex inclusion chain of the point set V which is an ordering $\mathcal{V} = (v_1, v_2, ..., v_n)$ of the points of V such that, for every $i \in \{2, ..., n\}$, v_i does not belong to the convex hull $conv(S_{i-1})$ (with $S_i = \{v_1, ..., v_i\}$, for all $i \in \{1, ..., n\}$). The set of k-sets of the convex inclusion chain $\mathcal{V}$ is then the set of distinct k-sets of $S_{k+1}, S_{k+2}, ..., S_n$.

The main result of this paper is that the number of k-sets of a convex inclusion chain of V is an invariant of the set V, that is, it does not depend on the choice of the convex inclusion chain. To prove this result, we use the notion of k-set polygon introduced by Edelsbrunner, Valtr, and Welzl [6]: the k-set polygon $g^k(V)$ of V is the convex hull of the centroids of all the subsets of k elements of V. Andrzejak and Fukuda [1] have shown that the vertices of this k-set polygon are the centroids of the k-sets of V. Thus, counting the number of k-sets comes to count the number of vertices or edges of the k-set polygon. In particular, we show that, given a point v not belonging to $conv(V)$, the edges of $g^k(V)$ that are not edges of $g^k(V \cup \{v\})$ form a connected polygonal line on the boundary $\delta(g^k(V))$ of $g^k(V)$. This generalizes a result which is well known in the case of convex hulls, that is, for $k = 1$. Using this result we show that:

Theorem 1. *Any convex inclusion chain of a planar set V of n points admits $2kn - n - k^2 + 1 - \sum_{j=1}^{k-1} \gamma^j(V)$ k-sets (with $\sum_1^0 = 0$).*

Surprisingly, this number is independent of the choice of $\mathcal{V}$ and is also equal to the number of regions of the order-k Voronoi diagram of V (see Lee [7]).

The best worst-case algorithm to construct the k-sets of a set V of n points in the plane has been given by Cole, Sharir, and Yap and runs in $O(n \log n + \gamma^k(V) \log^2 k)$ time [4] (for bigger values of k this can be improved to $O(n \log n + \gamma^k(V) \log^{1+\varepsilon} n)$ [3]).

In the second part of this paper we give an algorithmic method to update the set of k-sets of V when a point v that does not belong to $conv(V)$ is added. We apply this result to the on-line construction of the k-set polygon of a simple polygonal line $\mathcal{V}$ in the particular case where no vertex of $\mathcal{V}$ belongs to the convex hull of its predecessors on $\mathcal{V}$. This algorithm generalizes Melkman's on-line algorithm which constructs the convex hull of a simple polygonal line in linear time [8]. We show that:

Theorem 3. *The k-set polygon of the polygonal line $\mathcal{V}$ can be constructed on-line in $O(k(n-k) \log^2 k)$ time.*

The cost per created k-set is $O(\log^2 k)$, the same as in the algorithm of Cole, Sharir, and Yap [4].

2 Counting k-Sets of Convex Inclusion Chains

Throughout this paper we will consider V to be a finite set of $|V| = n$ points in the Euclidean plane such that $n \geq 2$ and no 3 points of V are colinear. k will be an integer of $\{1, ..., n-1\}$. The aim of this section is to count the number of k-sets of a convex inclusion chain of V. To this end we will use the boundary $\delta(g^k(V))$ of the k-set polygon of V. This boundary is considered to be oriented in counter clockwise direction. Moreover, given two points s and t of V, we denote by st the closed oriented segment with endpoints s and t, by (st) the oriented straight line generated by st, and by $(st)^+$ (resp. $(st)^-$) the open half plane on the left (resp. right) of (st). $\overline{(st)^+}$ and $\overline{(st)^-}$ denote the closure of $(st)^+$ and $(st)^-$.

Let us first recall two important properties of the vertices and edges of k-set polygons given by Andrzejak and Fukuda [1], and by Andrzejak and Welzl [2] (see Fig. 1 for an illustration).

Proposition 1. *The centroid $g(T)$ of a subset T of k points of V is a vertex of $g^k(V)$ if and only if T is a k-set of V. Moreover, the centroids of distinct k-sets are distinct vertices.*

Proposition 2. *T and T' are two k-sets of V such that $g(T)g(T')$ is an oriented edge of $g^k(V)$ if and only if there exist two points s and t of V and a subset P of $k-1$ points of V such that $T = P \cup \{s\}$, $T' = P \cup \{t\}$, and $V \cap (st)^- = P$.*

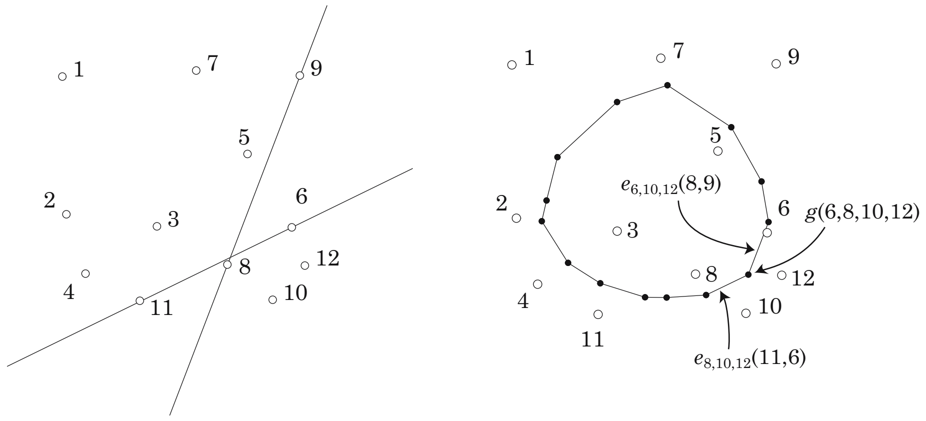


Fig. 1. A set of 12 points and its 4-set polygon

From now on, any oriented edge $g(P \cup \{s\})g(P \cup \{t\})$ of $g^k(V)$ will be denoted by $e_P(s,t)$.

Notice that, in the particular case where $k = 1$, $g^k(V)$ is the convex hull of V and its edges are of the form $e_\emptyset(s,t)$. When V is reduced to two points s and t, $g^1(V)$ admits exactly two oriented edges $e_\emptyset(s,t) = st$ and $e_\emptyset(t,s) = ts$.

We now characterize the edges of the k-set polygon that have to be created and those that have to be removed, when a new point is added (see Fig. 2). The following lemmas can easily be deduced from Proposition 2:

Lemma 1. *If $k < n-1$ then, for any subset S of V such that $k < |S| < n$ and for any point v of $V \setminus S$, an edge $e_P(s,t)$ of $g^k(S)$ is also an edge of $g^k(S \cup \{v\})$ if and only if $v \in (st)^+$.*

Lemma 2. *For any subset S of V such that $k \leq |S| < n$ and for any point v of $V \setminus S$,*

(i) an edge $e_P(s,t)$ of $g^k(S \cup \{v\})$ is not an edge of $g^k(S)$ if and only if $v \in P \cup \{s,t\}$,

(ii) if $v \notin conv(S)$, $g^k(S \cup \{v\})$ admits one and only one edge of the form $e_P(s,t)$ with $s = v$ (resp. $t = v$).

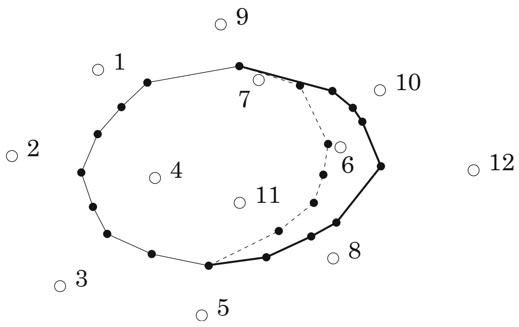


Fig. 2. The 4-set polygon of $S = \{1, ..., 11\}$ and the 4-set polygon of $S \cup \{12\}$. The edges of $g^k(S)$ that are not edges of $g^k(S \cup \{12\})$ are in dashed lines and the edges of $g^k(S \cup \{12\})$ that are not edges of $g^k(S)$ are in bold lines.

Note that, in Lemma 2, the definition of the k-set polygon has implicitly be extended to the case $k = |S|$. In this case, $g^k(S)$ is a unique point of the plane (the centroid of S) and, therefore, it admits no edge. This extended definition will help to simplify the proof of Proposition 4.

Proposition 3. *If $k < n - 1$ then, for any subset S of V such that $k < |S| < n$ and for any point v of $V \setminus conv(S)$,*

(i) the edges of $g^k(S \cup \{v\})$ that are not edges of $g^k(S)$ form an open connected polygonal line with at least two edges, whose first (resp. last) edge in counter clockwise direction is the unique edge of $g^k(S \cup \{v\})$ of the form $e_P(s,t)$ with $t = v$ (resp. $s = v$).

(ii) the edges of $g^k(S)$ that are not edges of $g^k(S \cup \{v\})$ form an open connected and non empty polygonal line.

Proof. (i) From Lemma 2, the set $\mathcal{C}$ of edges of $g^k(S \cup \{v\})$ that are not edges of $g^k(S)$ admits at least two edges. It can also be shown that at least one edge of $g^k(S)$ is an edge of $g^k(S \cup \{v\})$ too. Thus, $\mathcal{C}$ admits at least one edge $e_P(s,t)$ whose first endpoint is a vertex of $g^k(S)$, i.e. $v \notin P \cup \{s\}$. Hence, from Lemma 2, $t = v$ and $e_P(s,t)$ is the only edge of $g^k(S \cup \{v\})$ of the form $e_P(s,v)$. In the same way, there is a unique edge of $\mathcal{C}$ whose second endpoint is a vertex of $g^k(S)$ and this edge is of the form $e_P(v,t)$. It follows that $\mathcal{C}$ is an open connected polygonal line whose first (resp. last) edge in counter clockwise direction is of the form $e_P(s,v)$ (resp. $e_P(v,t)$).

(ii) Straightforward from (i). $\square$

This proposition generalizes a result which is well known in the case $k = 1$: The edges of a convex hull that are visible from a point outside of the hull form an open connected and non empty polygonal line. Moreover, if we want to update the convex hull after the insertion of such a point, two new edges have to be created. This means that the incremental construction of the convex hull of n points, in such a way that every newly inserted point does not belong to the current convex hull, constructs always $2(n-1)$ edges (two per inserted point except for the first one). We now generalize this last result for $k \neq 1$.

Let $(v_1, v_2, ..., v_n)$ be a convex inclusion chain of V, that is, an ordering of the points of V such that, for every $i \in \{1, ..., n-1\}$, $v_{i+1} \notin conv(S_i)$ (with $S_i = \{v_1, ..., v_i\}$, for every $i \in \{1, ..., n\}$).

For every $k \in \{1, ..., n-1\}$ and for every $i \in \{k+1, ..., n\}$, let c_i^k denote the number of edges of $g^k(S_i)$ that are not edges of $g^k(S_{i-1})$, i.e. the number of edges to create while constructing the k-set polygon of $S_i = S_{i-1} \cup v_i$ from the k-set polygon of S_{i-1}. Since the number of edges of $g^k(S_k)$ is zero, $c^k = \sum_{i=k+1}^{n} c_i^k$ is the total number of edges to be created by an algorithm that incrementally constructs $g^k(V)$ by successively determining $g^k(S_k)$, $g^k(S_{k+1})$, ..., $g^k(S_n)$.

From Proposition 1, for every $j \in \{1, ..., n-1\}$, the number of edges (i.e. the number of vertices) of the j-set-polygon of V is equal to the number $\gamma^j(V)$ of j-sets of V.

Proposition 4. *$c^1 = 2(n-1)$ and $c^k = k(2n-k-1) - \sum_{j=1}^{k-1} \gamma^j(V)$ if $1 < k < n$.*

Proof. From Lemma 2, for every $i \in \{k+1, ..., n\}$, $g^k(S_{i-1} \cup \{v_i\})$ admits at least two edges that are not edges of $g^k(S_{i-1})$. These two edges are of the form $e_Q(v_i, t)$ and $e_P(s, v_i)$. All other edges of $g^k(S_{i-1} \cup \{v_i\})$ that are not edges of $g^k(S_{i-1})$ are of the form $e_{P'}(s', t')$ with $v_i \in P'$. If $k = 1$, no such other edge exists since $P = \emptyset$. Thus $c_i^1 = 2$, for every $i \in \{2, ..., n\}$, and

$$c^1 = \sum_{i=2}^{n} 2 = 2(n-1) \ .$$

If $k \in \{2, ..., n-1\}$, from Lemma 1, $e_{P'}(s', t')$ is an edge of $g^k(S_{i-1} \cup \{v_i\})$ with $v_i \in P'$ if and only if $e_{P' \setminus \{v_i\}}(s', t')$ is an edge of $g^{k-1}(S_{i-1})$ and is not an edge of $g^{k-1}(S_{i-1} \cup \{v_i\})$. Thus, denoting by d_i^{k-1} the number of edges of $g^{k-1}(S_{i-1})$ that are not edges of $g^{k-1}(S_i)$, we have $c_i^k = 2 + d_i^{k-1}$. It follows that

$$c^k = \sum_{i=k+1}^{n} c_i^k = 2(n-k) + \sum_{i=k+1}^{n} d_i^{k-1} \ .$$

Now, since the number of edges of $g^{k-1}(S_{k-1})$ is zero, we have $d_k^{k-1} = 0$ and $\sum_{i=k+1}^{n} d_i^{k-1}$ is the total number of edges to be deleted by an algorithm that incrementally constructs $g^{k-1}(V)$ by successively determining $g^{k-1}(S_{k-1})$, $g^{k-1}(S_k)$, ..., $g^{k-1}(S_n)$. Thus

$$\sum_{i=k+1}^{n} d_i^{k-1} = c^{k-1} - \gamma^{k-1}(V)$$

and

$$c^k = 2(n-k) + c^{k-1} - \gamma^{k-1}(V) \ .$$

Solving this induction relation we get

$$c^k = (k-1)(2n-k-2) + c^1 - \sum_{j=1}^{k-1} \gamma^j(V) = k(2n-k-1) - \sum_{j=1}^{k-1} \gamma^j(V) \ .$$

□

The result of this proposition is somewhat surprising since it shows that the number of edges that have to be created for the incremental construction of a k-set polygon does not depend on the order in which the points are treated, provided that every new inserted point does not belong to the convex hull of the previously inserted ones. In addition, since $\sum_{j=1}^{k-1} \gamma^j(V)$ is the number of $(\leq (k-1))$-sets of V and since this number is known to be bounded by $(k-1)n$ (see [11]), it follows that:

Corollary 1. *Any algorithm that incrementally constructs the k-set polygon of n points, so that no point belongs to the convex hull of the points inserted before him, has to create $\Theta(k(n-k))$ edges.*

Now, it is easy to find the number of k-sets of a convex inclusion chain of V:

Theorem 1. *Any convex inclusion chain of a planar set V of n points admits $2kn - n - k^2 + 1 - \sum_{j=1}^{k-1} \gamma^j(V)$ k-sets (with $\sum_1^0 = 0$).*

Proof. Taking the previous notations, if $\mathcal{V} = (v_1, v_2, ..., v_n)$ is a convex inclusion chain of V, the number of k-sets of $\mathcal{V}$ is equal to the number of distinct k-set polygon vertices created by an incremental algorithm that successively constructs $g^k(S_{k+1})$, ..., $g^k(S_n)$. The number of vertices of $g^k(S_{k+1})$ is equal to the number c^k_{k+1} of its edges. Moreover, from Proposition 3, for every $i \in \{k+2, ..., n\}$, the edges of $g^k(S_i)$ that are not edges of $g^k(S_{i-1})$ form an open connected and non empty polygonal line. Thus, the number of vertices of this line that are not vertices of $g^k(S_{i-1})$ is $c^k_i - 1$, where c^k_i is the number of edges of the line. It follows that the number of k-sets of $\mathcal{V}$ is $c^k_{k+1} + \sum_{i=k+2}^{n} (c^k_i - 1)$, that is, from Proposition 4, $2kn - n - k^2 + 1 - \sum_{j=1}^{k-1} \gamma^j(V)$. □

According to this theorem, the number of k-sets of a convex inclusion chain of V only depends on the set V and not on the chosen chain. An even more intriguing consequence of the theorem arises from its connection with order-k Voronoi diagrams. The order-k Voronoi diagram of V is a partition of the plane in regions which are the set of points in the plane having the same k nearest neigbours in V. Lee [7] has shown that, if no four points of V are cocircular, the order-k Voronoi diagram of V admits $2kn - n - k^2 + 1 - \sum_{j=1}^{k-1} \gamma^j(V)$ regions; the same number as the one found in Theorem 1. Since a subset of k points of V generates an order-k Voronoi region if and only if it can be separated from the remaining points by a circle, it follows that:

Corollary 2. *Given a set V of points in the plane, no three of them being colinear and no four of them being cocircular, the number of k-sets of a convex inclusion chain of V is equal to the number of subsets of k points of V that can be separated from the remaining by a circle.*

3 Constructing k-Sets of Convex Inclusion Chains

In this section we will consider the construction of the k-sets of a convex inclusion chain of V. In particular, we will show how to update the set of k-sets of a subset S of V when a new point, that does not belong to $conv(S)$, is added. As in the previous section, we will use the k-set polygon of V as a powerful tool.

For $k \in \{1, \dots, |V|-2\}$, for S a subset of V such that $k < |S| < n$, and for v a point of $V \setminus conv(S)$, we set the following notations:

(i) Let $\mathcal{C}_{S,v}$ (resp. $\mathcal{D}_{S,v}$) denote the counter clockwise oriented polygonal line of edges of $g^k(S \cup \{v\})$ (resp. $g^k(S)$) that are not edges of $g^k(S)$ (resp. $g^k(S \cup \{v\})$).

(ii) Let $T_1, T_2, ..., T_m$ denote the k-sets of S such that $(g(T_1), g(T_2), ..., g(T_m))$ is the ordered sequence of vertices of $\mathcal{D}_{S,v}$ (including its two endpoints).

(iii) For every $i \in \{1, ..., m\}$, let $e_{P_i}(s_i, t_i)$ denote the oriented edge of $g^k(S)$ whose second endpoint is $g(T_i)$ and let $e_{P_{m+1}}(s_{m+1}, t_{m+1})$ denote the oriented edge whose first endpoint is $g(T_m)$.

(iv) Set $\alpha_1 = \omega_m = v$ and, for every $i \in \{2, ..., m\}$, set $\alpha_i = t_i$ and $\omega_{i-1} = s_i$ (see Fig. 3 for an illustration).

(v) For every $i \in \{1, ..., m\}$, if $\alpha_i \neq \omega_i$, $\varphi(T_i)$ denotes the oriented polygonal line that connects α_i to ω_i in counter clockwise direction on $\delta(conv(T_i \cup \{v\}))$ and, if $\alpha_i = \omega_i$, set $\varphi(T_i) = \alpha_i$.

(vi) For every $i \in \{1, ..., m\}$, let $\mathcal{H}_i$ denote the homothety of center $g(T_i \cup \{v\})$ and ratio $-\frac{1}{k}$, that is, for any point x in the plane, $\mathcal{H}_i(x) = g((T_i \cup \{v\}) \setminus \{x\})$.

Notation (v) makes sense since, for every $i \in \{1, ..., m\}$, α_i and ω_i are vertices of $conv(T_i \cup \{v\})$. Indeed, $\alpha_1 = \omega_m = v$ can be separated from S by a straight line and is thus a vertex of $conv(T_1 \cup \{v\})$ and of $conv(T_m \cup \{v\})$. Moreover, from Proposition 2, for every $i \in \{2, ..., m\}$, $T_i \setminus t_i = P_i \subset (s_i t_i)^-$ and, from Lemma 1, $v \in (s_i t_i)^-$. Thus, $\alpha_i = t_i$ is a vertex of $conv(T_i \cup \{v\})$, for all $i \in \{2, ..., m\}$. In the same way, $\omega_i = s_{i+1}$ is a vertex of $conv(T_i \cup \{v\})$, for all $i \in \{1, ..., m-1\}$.

We now show that every vertex $g(T_i)$ of $\mathcal{D}_{S,v}$ can be associated to a subset of $\mathcal{C}_{S,v}$.

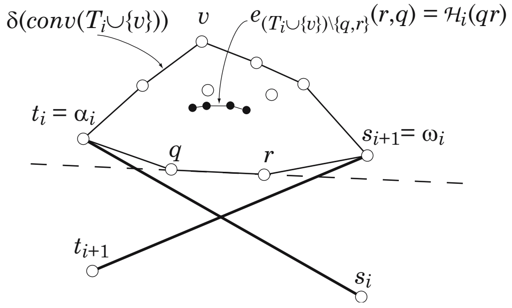

$\delta(conv(T_i \cup \{v\}))$
v
$e_{(T_i \cup \{v\}) \setminus \{q,r\}}(r,q) = \mathcal{H}_i(qr)$
$t_i = \alpha_i$
$s_{i+1} = \omega_i$
q
r
t_{i+1}
s_i

Fig. 3. $s_i t_i$ and $s_{i+1} t_{i+1}$ are such that $e_{P_i}(s_i, t_i)$ and $e_{P_{i+1}}(s_{i+1}, t_{i+1})$ are the two consecutive edges of $\mathcal{D}_{S,v}$ sharing the vertex $g(T_i)$

Lemma 3. *For every $i \in \{1, ..., m\}$ such that $\varphi(T_i)$ is not reduced to a single point and for every oriented edge qr of $\varphi(T_i)$, $\mathcal{H}_i(qr)$ is the edge $e_{T_i \cup \{v\} \setminus \{q,r\}}(r, q)$ of $\mathcal{C}_{S,v}$.*

Proof. (i) We first show that, if $i \in \{2, ..., m-1\}$, then $S \setminus T_i \subset (qr)^-$. Since $g(T_i)$ is the vertex shared by the edges $e_{P_i}(s_i, t_i)$ and $e_{P_{i+1}}(s_{i+1}, t_{i+1})$, we have $g(T_i) = g(P_i \cup \{t_i\}) = g(P_{i+1} \cup \{s_{i+1}\})$ and, from Proposition 1, $T_i = P_i \cup \{t_i\} = P_{i+1} \cup \{s_{i+1}\}$. From Proposition 2, it follows that $T_i \subset \overline{(s_i t_i)^-} \cap \overline{(s_{i+1} t_{i+1})^-}$ and that $S \setminus T_i \subset \overline{(s_i t_i)^+} \cap \overline{(s_{i+1} t_{i+1})^+}$. Thus, $(s_i t_i)$ and $(s_{i+1} t_{i+1})$ are the common tangents of $conv(T_i)$ and of $conv(S \setminus T_i)$ such that $conv(T_i)$ and $conv(S \setminus T_i)$ are on both sides of these tangents (see Fig. 3). Moreover, since $\{t_i, s_{i+1}\} \subset T_i$ and $\{s_i, t_{i+1}\} \subset S \setminus T_i$, the intersection point of the segments $s_i t_i$ and $s_{i+1} t_{i+1}$ is either the point $t_i = s_{i+1}$ or a point of $(s_{i+1} t_i)^+$. Since $\varphi(T_i)$ is not reduced to a single point, $t_i \neq s_{i+1}$ and, since $v \notin conv(S)$, it follows from Lemma 1 that $v \in (s_i t_i)^- \cap (s_{i+1} t_{i+1})^- \cap (s_{i+1} t_i)^-$. Thus, $(s_i t_i)$ and $(s_{i+1} t_{i+1})$ are also common tangents of $conv(T_i \cup \{v\})$ and of $conv(S \setminus T_i)$. The edges of $\varphi(T_i)$ are then the edges of $conv(T_i)$ included in $(s_{i+1} t_i)^+$ and the slopes of the oriented straight lines generated by such edges are comprised between the slopes of $(t_i s_i)$ and $(t_{i+1} s_{i+1})$. Thus, the edges of $\varphi(T_i)$ are the edges of $conv(T_i \cup \{v\})$ visible from every point of $\overline{(s_i t_i)^+} \cap \overline{(s_{i+1} t_{i+1})^+}$ and, since $S \setminus T_i \subset \overline{(s_i t_i)^+} \cap \overline{(s_{i+1} t_{i+1})^+}$, it follows that any edge qr of $\varphi(T_i)$ is such that $S \setminus T_i \subset (qr)^-$.

In a similar way, we can show that $S \setminus T_1 \subset (qr)^-$ and $S \setminus T_m \subset (qr)^-$.

(ii) Since every edge qr of $\varphi(T_i)$ is an edge of $conv(T_i \cup \{v\})$, for every $i \in \{1, ..., m\}$, we have $(T_i \cup \{v\}) \setminus \{q, r\} \subset (qr)^+$. Moreover, since $|(T_i \cup \{v\}) \setminus \{q, r\}| = k - 1$, it follows from (i) and from Proposition 2 that $e_{(T_i \cup \{v\}) \setminus \{q,r\}}(r, q)$ is an edge of $g^k(S \cup \{v\})$ and, from Lemma 2, that this edge belongs to $\mathcal{C}_{S,v}$. Moreover, the endpoints $g((T_i \cup \{v\}) \setminus \{q\})$ and $g((T_i \cup \{v\}) \setminus \{r\})$ of this edge are the respective images of q and r by the homothety $\mathcal{H}_i$ of center $g(T_i \cup \{v\})$ and ratio $-\frac{1}{k}$. Thus $e_{(T_i \cup \{v\}) \setminus \{q,r\}}(r, q) = \mathcal{H}_i(qr)$. □

And thus, the complete characterization of the line $\mathcal{C}_{S,v}$:

Theorem 2. *$\mathcal{C}_{S,v}$ is the sequence of polygonal lines $(\mathcal{H}_1(\varphi(T_1)), ..., \mathcal{H}_m(\varphi(T_m)))$.*

Proof. (i) From Lemma 3, for every $i \in \{1, ..., m\}$, if $\varphi(T_i)$ admits at least one edge, $\mathcal{H}_i(\varphi(T_i))$ is a connected polygonal line included in $\mathcal{C}_{S,v}$. Moreover, for every $j \in \{1, ..., m\}$ such that $j \neq i$, we have $T_i \neq T_j$ and thus, for every edge $q_i r_i$ of $\varphi(T_i)$ and for every edge $q_j r_j$ of $\varphi(T_j)$, $\mathcal{H}_i(q_i r_i) = e_{(T_i \cup \{v\}) \setminus \{q_i, r_i\}}(r_i, q_i)$ and $\mathcal{H}_j(q_j r_j) = e_{(T_j \cup \{v\}) \setminus \{q_j, r_j\}}(r_j, q_j)$ are distinct edges of $\mathcal{C}_{S,v}$. Hence, $\mathcal{H}_i(\varphi(T_i))$ and $\mathcal{H}_j(\varphi(T_j))$ share no edge.

(ii) Let us now show that all the polygonal lines $\mathcal{H}_i(\varphi(T_i))$, $i \in \{1, ..., m\}$, fill $\mathcal{C}_{S,v}$. By definition, the first edge of $\varphi(T_1)$ connects v to a point r of T_1. This edge always exists since the two endpoints $\alpha_1 = v$ and $\omega_1 = s_2$ of $\varphi(T_1)$ are distinct. From Lemma 3 and Proposition 3, $\mathcal{H}_1(vr) = e_{T_1 \setminus \{r\}}(r, v)$ is then the first edge of $\mathcal{C}_{S,v}$ and $\mathcal{H}_1(\varphi(T_1))$ is an initial subsequence of $\mathcal{C}_{S,v}$. In the same way, $\mathcal{H}_m(\varphi(T_m))$ is a final subsequence of $\mathcal{C}_{S,v}$. Moreover, for all $i \in \{1, ..., m-1\}$, $\omega_i = s_{i+1}$ and $\alpha_{i+1} = t_{i+1}$, that is $\mathcal{H}_i(\omega_i) = g((T_i \cup \{v\}) \setminus \{s_{i+1}\})$ and $\mathcal{H}_{i+1}(\alpha_{i+1}) = g((T_{i+1} \cup$

$\{v\}) \setminus \{t_{i+1}\})$. From Proposition 2, $T_i \setminus \{s_{i+1}\} = T_{i+1} \setminus \{t_{i+1}\}$ and it follows that $\mathcal{H}_i(\omega_i) = \mathcal{H}_{i+1}(\alpha_{i+1})$. Finally, $\mathcal{C}_{S,v} = (\mathcal{H}_1(\varphi(T_1)), ..., \mathcal{H}_m(\varphi(T_m)))$. □

The result of this theorem can now be used to develop an algorithm that updates the k-set polygon of S when v is added. Let us first describe the data structure to implement. The boundary of the k-set polygon of S can be stored in a circular list L whose elements represent the edges of $g^k(S)$. To any element e of L which represents an edge $e_P(s,t)$ of $g^k(S)$ are associated the two elements of L that represent the predecessor and the successor of $e_P(s,t)$ on $\delta(g^k(S))$, as well as the two points s and t of S. Note that, from Proposition 2, the k-sets defining two consecutive vertices of $g^k(S)$ differ from each other by one site and thus it suffices to know one k-set T of S and one edge with endpoint $g(T)$ to be able to generate the whole k-sets of S while traversing L. It follows that a k-set polygon with c edges can be stored in a data structure of size $O(c+k)$ and thus provides a compact way to encode the k-sets of a given point set.

In our algorithm we also use a data structure CH that allows dynamic convex hull maintenance. Using results given by Overmars and van Leeuwen [10] (see also Overmars [9]), this structure needs $O(h)$ size to store the convex hull of h points of the plane, allows to get the predeccessor and the successor of any edge in constant time, and can be updated in $O(\log^2 h)$ time after inserting or deleting a point.

For any polygonal line $\mathcal{P}$, let now $|\mathcal{P}|$ denote the number of vertices of $\mathcal{P}$.

Proposition 5. *The edges of $\mathcal{D}_{S,v}$ can be removed from L and the edges of $\mathcal{C}_{S,v}$ inserted in L in $O(|\mathcal{D}_{S,v}| \log^2 k + |\mathcal{C}_{S,v}|)$ time provided that one edge e of L belonging to $\mathcal{D}_{S,v}$ is given, and that the convex hull of one k-set T of S whose centroid is an endpoint of e is stored in CH.*

Proof. From Theorem 2, determining $\mathcal{C}_{S,v}$ comes, for every $i \in \{1, ..., m\}$, to determine $\mathcal{H}_i(\varphi(T_i))$ where $\varphi(T_i)$ is a connected subset of $conv(T_i \cup \{v\})$. Suppose that an edge e of L belonging to $\mathcal{D}_{S,v}$ is given, and that the convex hull of a k-set T of S whose centroid is an endpoint of e is stored in CH. We first show that $conv(T_1 \cup \{v\})$ can be determined from $conv(T)$ in $O(|\mathcal{D}_{S,v}| \log^2 k)$ time. From Lemma 1 and Proposition 3, $\mathcal{D}_{S,v}$ is the polygonal line formed by the edges $e_P(s,t)$ of $g^k(S)$ such that v is on the right of (st). Since the points s and t are associated to the edge $e_P(s,t)$ in L, since constant time is needed to test on which side of (st) v lies, and since the neighbours of any edge in L can also be obtained in constant time, it follows that $e_{P_1}(s_1, t_1)$ can be found, starting from e, in $O(|\mathcal{D}_{S,v}|)$ time. Moreover, from Proposition 2, $T_{i-1} = (T_i \setminus \{t_i\}) \cup \{s_i\}$, for every $i \in \{2, ..., m\}$. Hence $conv(T_{i-1})$ can be computed from $conv(T_i)$ in $O(\log^2 k)$ time. Thus, while searching $e_{P_1}(s_1, t_1)$, $conv(T)$ can be replaced by $conv(T_1)$ in CH in $O(|\mathcal{D}_{S,v}| \log^2 k)$ time and $conv(T_1 \cup \{v\})$ can then be deduced in $O(\log^2 k)$ time. Now, the polygonal line $\varphi(T_1)$ which connects $\alpha_1 = v$ and $\omega_1 = s_2$ on $\delta(conv(T_1 \cup \{v\}))$ can be reported in $O(|\varphi(T_1)|)$ time. From Lemma 3, for every edge qr of $\varphi(T_1)$, the edge $\mathcal{H}_1(qr) = e_{(T_1 \cup \{v\}) \setminus \{q,r\}}(r,q)$ is an edge of $\mathcal{C}_{S,v}$. This comes to insert a new edge in L to which are associated its two neigbours in L as well as the points r and q. Since the edges of $\mathcal{H}_1(\varphi(T_1))$

appear in L in the same order as their corresponding edges on $\varphi(T_1)$, it follows that $\mathcal{H}_1(\varphi(T_1))$ can be inserted in L in $O(|\varphi(T_1)|)$ time. In the same way, for every $i \in \{2, ..., m\}$, $T_i = (T_{i-1} \setminus \{s_i\}) \cup \{t_i\}$ and thus $conv(T_i \cup \{v\})$ can be computed from $conv(T_{i-1} \cup \{v\}))$ in $O(\log^2 k)$ time. $\varphi(T_i)$, which connects $\alpha_i = t_i$ and ω_i, can then be reported in $O(|\varphi(T_i)|)$ time and $\mathcal{H}_i(\varphi(T_i))$ can also be inserted in L in $O(|\varphi(T_i)|)$ time (note that, from Theorem 2, $\mathcal{H}_i(\varphi(T_i))$ follows $\mathcal{H}_{i-1}(\varphi(T_{i-1}))$ in L). Finally, L can be updated after the insertion of v in total $O(|\mathcal{D}_{S,v}| \log^2 k + \sum_{i=1}^{m} |\varphi(T_i)|)$, that is, $O(|\mathcal{D}_{S,v}| \log^2 k + |\mathcal{C}_{S,v}|)$ time. □

Remark 1. Notice that at the end of the algorithm described by Proposition 5, the data structure CH contains the convex hull of T_m, with $g(T_m)$ a common endpoint of $\mathcal{D}_{S,v}$ and $\mathcal{C}_{S,v}$. Moreover, the edge of $\mathcal{C}_{S,v}$ with endpoint $g(T_m)$ is the last edge inserted in L and therefore it can be easily maintained.

We will now show how Proposition 5 can be applied to the on-line construction of the k-set polygon of a planar simple polygonal line $\mathcal{V} = (v_1, v_2, ..., v_n)$ of n vertices which is such that, for every $i \in \{2, ..., n\}$, $v_i \notin conv(S_{i-1})$ (with $S_i = \{v_1, ..., v_i\}$ for every $i \in \{1, ..., n\}$).

Theorem 3. *The k-set polygon of the polygonal line $\mathcal{V}$ can be constructed on-line in $O(k(n-k)\log^2 k)$ time.*

Proof. (i) We first show that the k-set polygon of $S_{k+1} = \{v_1, ..., v_{k+1}\}$ can be computed in $O(k)$ time. From Proposition 2, every edge of $g^k(S_{k+1})$ is of the form $e_{S_{k+1} \setminus \{s,t\}}(s,t)$ where ts is an edge of $conv(S_{k+1})$. Conversely, if ts is an edge of $conv(S_{k+1})$, then $e_{S_{k+1} \setminus \{s,t\}}(s,t)$ is an edge of $g^k(S_{k+1})$. In addition, if $e_{S_{k+1} \setminus \{s',t'\}}(s',t')$ is the successor of $e_{S_{k+1} \setminus \{s,t\}}(s,t)$ on $\delta(g^k(S_{k+1}))$, then $(S_{k+1} \setminus \{s,t\}) \cup \{t\} = (S_{k+1} \setminus \{s',t'\}) \cup \{s'\}$, that is, $s = t'$. ts and $t's'$ are therefore two consecutive edges of $conv(S_{k+1})$ and it follows that constructing $g^k(S_{k+1})$ comes to construct $conv(S_{k+1})$. The convex hull of a simple polygonal line of $k+1$ vertices can be constructed on-line in $O(k)$ time using Melkman's algorithm [8].

Moreover, let $e = e_{S_{k+1} \setminus \{s,t\}}(s,t)$ be an edge of $g^k(S_{k+1})$. Setting $T = S_{k+1} \setminus \{s\}$, $g(T)$ is an endpoint of e and $conv(T)$ can be stored in the data structure CH in $O(k \log^2 k)$ time.

(ii) We now show that, for every $i \in \{k+2, ..., n\}$, $g^k(S_i)$ can be computed from $g^k(S_{i-1})$ in $O((|\mathcal{D}_{S_{i-1},v_i}| + |\mathcal{C}_{S_{i-2},v_{i-1}}|) \log^2 k + |\mathcal{C}_{S_{i-1},v_i}|)$ time (here $\mathcal{C}_{S_k,v_{k+1}}$ denotes the boundary of $g^k(S_{k+1})$).

(ii.1) We first prove that at least one edge of $\mathcal{D}_{S_{i-1},v_i}$ is also an edge of $\mathcal{C}_{S_{i-2},v_{i-1}}$. From the definition of $\mathcal{V}$, v_{i-1} is a vertex of $conv(S_{i-1})$ visible from v_i. Thus, there exists an oriented straight line Δ passing through v_{i-1}, that is not parallel to any straight line passing through any two points of S_{i-1}, and such that $conv(S_{i-1}) \subset \overline{\Delta^+}$ and $v_i \in \Delta^-$. Let Δ' be a straight line parallel to Δ, oriented in the same direction as Δ and such that $|\Delta'^- \cap S_{i-1}| = k$. Let $U = \Delta'^- \cap S_{i-1}$. Let (st) and $(s't')$ be the oriented straight lines tangent to both $conv(U)$ and $conv(S_{i-1} \setminus U)$ such that $\{s', t\} \subseteq U$, $conv(U) \subset \overline{(st)^-}$, and $conv(S_{i-1} \setminus U) \subset \overline{(st)^+}$ (resp. $conv(U) \subset \overline{(s't')^-}$, and $conv(S_{i-1} \setminus U) \subset \overline{(s't')^+}$). Thus, from Proposition 2 and Lemma 2, $e_{U \setminus \{t\}}(s,t)$ and $e_{U \setminus \{s'\}}(s',t')$ are edges of $g^k(S_{i-1})$ that belong to $\mathcal{C}_{S_{i-2},v_{i-1}}$, since $v_{i-1} \in U$. Now, v_i cannot belong to both $(st)^+$ and $(s't')^+$ and, from Lemma 1, at least one of $e_{U\setminus\{t\}}(s,t)$ and $e_{U\setminus\{s'\}}(s',t')$ belongs to $\mathcal{D}_{S_{i-1},v_i}$. Hence, at least one edge of $\mathcal{D}_{S_{i-1},v_i}$ is also an edge of $\mathcal{C}_{S_{i-2},v_{i-1}}$.

(ii.2) Now, from (i) and from Remark 1, after the construction of $g^k(S_{i-1})$, an edge e of $\mathcal{C}_{S_{i-2},v_{i-1}}$ is given and the convex hull of a k-set T whose centroid is a vertex of e is known. From (ii.1), an edge e' of $\mathcal{D}_{S_{i-1},v_i}$ can then be found, starting from e, in $O(|\mathcal{C}_{S_{i-2},v_{i-1}}|)$ time, as in the proof of Proposition 5. The same, the convex hull of a k-set T' whose centroid is a vertex of e' can be constructed, starting from *conv*(T), in $O(|\mathcal{C}_{S_{i-2},v_{i-1}}|\log^2 k)$ time. Thus, from Proposition 5, for every $i \in \{k+2, ..., n\}$, $g^k(S_i)$ can be constructed from $g^k(S_{i-1})$ in $O((|\mathcal{D}_{S_{i-1},v_i}| + |\mathcal{C}_{S_{i-2},v_{i-1}}|)\log^2 k + |\mathcal{C}_{S_{i-1},v_i}|)$ time.

(iii) It follows from (i) and (ii) that the k-set polygon of $\mathcal{V}$ can be constructed on-line in $O(k\log^2 k + \sum_{i=k+2}^{n}((|\mathcal{D}_{S_{i-1},v_i}| + |\mathcal{C}_{S_{i-2},v_{i-1}}|)\log^2 k + |\mathcal{C}_{S_{i-1},v_i}|))$ time. By setting, as in section 1, $c^k = \sum_{i=k+1}^{n} |\mathcal{C}_{S_{i-1},v_i}|$, we have $\sum_{i=k+2}^{n}((|\mathcal{D}_{S_{i-1},v_i}| + |\mathcal{C}_{S_{i-2},v_{i-1}}|)\log^2 k + |\mathcal{C}_{S_{i-1},v_i}|) \leq 2c^k \log^2 k + c^k$. From Proposition 4, c^k is in $O(k(n-k))$ and the time complexity of the algorithm is $O(k(n-k)\log^2 k)$. $\square$

From Corollary 1, any algorithm that incrementally constructs the k-set polygon of the polygonal line $\mathcal{V}$, has to generate $\Omega(k(n-k))$ edges. It follows that the time complexity of the above algorithm, per edge that has to be created, is $O(\log^2 k)$. This complexity can be compared to the one in the algorithm given by Cole, Sharir, and Yap [4] which constructs the set of k-sets of n points in the plane in $O(n\log n + c\log^2 k)$ time, where c is the total number of k-sets of the n points.

4 Conclusion

In this paper we have shown that all the convex inclusion chains of a given set V of points in the plane admit the same number of k-sets. This number is also equal to the number of regions of the order-k Voronoi diagram of V. Up to now we do not know any direct proof of this last result. Such a proof would provide a completely different way to count the number of regions of the order-k Voronoi diagrams in the plane. Studying these relations in higher dimensions would then be of great interest since the size of the order-k Voronoi diagrams is not known in dimension greater than two.

By using the properties of the k-set polygons, we have also given an algorithm to update the set of k-sets of V when a new point that does not belong to *conv*(V) is added. This algorithm has then be applied to the on-line construction of the k-sets of certain simple polygonal lines. The time complexity of both algorithms is $O(\log^2 k)$ per created edge. This factor comes from the use of the dynamic convex hull data structure of Overmars and van Leeuwen [10]. Chan [3] has given a data structure that allows dynamic maintenance of the convex hull of n points in $O(lg^{1+\varepsilon}n)$ amortized time. Using this data structure, the overall complexity of our second algorithm becomes $O(k(n-k)\,\mathrm{lg}^{1+\varepsilon} n)$, which is interesting for bigger values of k.

References

1. A. Andrzejak and K. Fukuda. Optimization over k-set polytopes and efficient k-set enumeration. In *Proc. 6th Workshop Algorithms Data Struct.*, volume 1663 of *Lecture Notes Comput. Sci.*, pages 1–12. Springer-Verlag, 1999.
2. A. Andrzejak and E. Welzl. In between k-sets, j-facets, and i-faces: (i, j)-partitions. *Discrete Comput. Geom.*, 29:105–131, 2003.
3. T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *J. ACM*, 48:1–12, 2001.
4. R. Cole, Micha Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16:61–77, 1987.
5. T. K. Dey. Improved bounds on planar k-sets and related problems. *Discrete Comput. Geom.*, 19:373–382, 1998.
6. H. Edelsbrunner, P. Valtr, and Emo Welzl. Cutting dense point sets in half. *Discrete Comput. Geom.*, 17:243–255, 1997.
7. D. T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, C-31:478–487, 1982.
8. A. Melkman. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.*, 25:11–12, 1987.
9. M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983.
10. M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.
11. G. W. Peck. On k-sets in the plane. *Discrete Math.*, 56:73–74, 1985.
12. G. Tóth. Point sets with many k-sets. *Discrete Comput. Geom.*, 26(2):187–194, 2001.

Toward the Eigenvalue Power Law[⋆]

Robert Elsässer

University of Paderborn
Institute for Computer Science
33102 Paderborn, Germany
elsa@upb.de

Abstract. Many graphs arising in various real world networks exhibit the so called "power law" behavior, i.e., the number of vertices of degree i is proportional to $i^{-\beta}$, where $\beta > 2$ is a constant (for most real world networks $\beta \leq 3$). Recently, Faloutsos et al. [18] conjectured a power law distribution for the eigenvalues of power law graphs. In this paper, we show that the eigenvalues of the Laplacian of certain random power law graphs are close to a power law distribution.

First we consider the generalized random graph model $G(\mathbf{d}) = (V, E)$, where $d = (d_1, \ldots, d_n)$ is a given sequence of expected degrees, and two nodes $v_i, v_j \in V$ share an edge in $G(\mathbf{d})$ with probability $p_{i,j} = d_i d_j / \sum_{k=1}^{n} d_k$, independently [9]. We show that if the degree sequence $\mathbf{d}$ follows a power law distribution, then some largest $\Theta(n^{1/\beta})$ eigenvalues of $L(\mathbf{d})$ are distributed according to the same power law, where $L(\mathbf{d})$ represents the Laplacian of $G(\mathbf{d})$. Furthermore, we determine for the case $\beta \in (2, 3)$ the number of Laplacian eigenvalues being larger than i, for any $i = \omega(1)$, and compute how many of them are in some range $(i, (1+\epsilon)i)$, where $i = \omega(1)$ and $\epsilon > 0$ is a constant. Please note that the previously described results are guaranteed with probability $1 - o(1/n)$.

We also analyze the eigenvalues of the Laplacian of certain dynamically constructed power law graphs defined in [2,3], and discuss the applicability of our methods in these graphs.

1 Introduction

Eigenvalues of graphs have a various range of graph theoretic applications including the estimation of expansion properties, bisection width, or distances in graphs (see e.g. [8]), and are useful for determining the behavior of certain algorithms in the field of low ranked matrix approximation [1], information retrieval [25], load balancing [14,15], and computer vision [19]. Of particular interest is the study of eigenvalues of graphs modeling large scale real world networks. In [2,3,18,24] it has been observed that in many real-world networks, including the Web, the Internet, telephone call graphs, and various social and biological networks, the degrees of the nodes have a so called power law distribution, i.e.,

[⋆] The research was performed while the author visited the Department of Mathematics at University of California, San Diego. Partly supported by the German Research Foundation under contract EL-399/1-1.

the fraction of vertices with degree d is proportional to $d^{-\beta}$, where $\beta > 2$ is a fixed constant (for most of the previously mentioned networks it holds that $\beta \in (2,3)$). Barabási and Albert suggested to model complex real world networks by the following dynamic random graph process: We start with a complete graph consisting of n_0 vertices, and then new nodes are added to the graph one at a time and joined to a fixed number $m < n_0$ of existing vertices, selected with probabilities proportional to their degrees [3]. In [5] it has been shown that the degree distribution of the graphs constructed by a similar method are close to a power law with $\beta = 3$, i.e., the fraction of vertices with degree d is proportional to d^{-3} for any $d \in \{m, \dots, n^{1/15}\}$.

Several other dynamic random graph models have been proposed and analyzed (e.g. [2,6,26]). Another approach is to model power law networks by a static random graph process. Clearly, the classical random graph model of [16,20], in which two nodes are connected with some predefined probability p, is not well suited to model such networks. Therefore, Chung and Lu proposed a more general random graph model with arbitrary degree distributions: For a sequence $\mathbf{d} = (d_1, \dots, d_n)$ let $G(\mathbf{d})$ be the graph in which edges are independently assigned to each pair of vertices (v_i, v_j) with probability $d_i d_j / \sum_{k=1}^{n} d_k$ [9]. If now the degree distribution $\mathbf{d}$ obeys a power law, then the resulting graph is well suited for modeling power law graphs.

Recently, several properties of these graphs have been analyzed. In [9], Chung and Lu determined the size and the volume of the connected components. In [10] they analyzed the distances between two nodes in such graphs. Mihail and Papadimitriou [27] considered the spectrum of the adjacency matrix of these graphs and showed that its eigenvalues are distributed according to a power law, if $\beta > 3$. In [11] Chung et al. concentrated on the case $\beta \in (2,3]$, and showed that if $\beta > 2.5$, then the k largest eigenvalues of the adjacency matrix of $G(\mathbf{d})$ follow a power law with exponent $2\beta - 1$, where it is assumed that the k^{th} largest degree is significantly larger than $\tilde{d} = \sum_{i=1}^{n} d_i^2 / \sum_{i+1}^{n} d_i$. If $\beta < 2.5$, then the largest eigenvalue is $\tilde{d}(1 + o(1))$.

Chung et al. also analyzed the eigenvalues of the normalized Laplacian of these graphs. In [12] they proved a very nice result which states that the distribution of the afore mentioned eigenvalues satisfy the semi-circle law under the condition that the minimum expected degree is much larger than the square root of the expected average degree.

We should mention here that the real world networks described before also possess other properties (e.g. exhibit high levels of clustering, cf. [28]) which are unspecific for the random graphs considered here. Although we do not deal with graphs which have the other properties, we hope the techniques and results stated in this paper might provide insights at a more general level, too.

Our main goal is to investigate the influence of the power law structure on the spectrum of graphs. Previous results stated strong relationships between the random power law structure and the eigenvalues of the normalized Laplacian or adjacency matrix. In this paper, we study the eigenvalues of the combinatorial Laplacian of random power law graphs.

First, we determine the largest $\sum_{j=\tau}^{d_{\max}} \frac{n}{j^{\beta}} \cdot (1+o(1))$ Laplacian eigenvalues of the $G(\mathbf{d})$, where $\tau = \left\lceil d_{\max}\left(1 - \sum_{k=d_{\min}}^{\infty} \frac{k^{-\beta}e^{-k}}{\sum_{k=d_{\min}}^{\infty} k^{-\beta}}\right)\right\rceil$, $\mathbf{d}$ obeys a power law distribution with exponent β, and $d_{\min}$ and $d_{\max}$ denote the smallest and largest degree in $G(\mathbf{d})$, respectively. Then, we focus on the distribution of all eigenvalues larger than $\omega(1)$, and show that it is close to a power law with the same exponent β. We also consider some dynamically constructed power law graphs, and discuss the applicability of our methods in these graphs.

The rest of the paper is organized in three sections. In Section 2 we state some auxiliary results needed in our proofs. In Section 3 we consider the Laplacian spectrum of the graphs described above, and prove the eigenvalue power law properties. In Section 4 we summarize the results and point to some open problems.

2 Preliminaries

The primary model for random graphs is the Erdős-Rényi model G_p, in which two vertices are connected by an edge with probability p, independently, for some given $p > 0$ [17]. In such random graphs the degrees of vertices all have the same expected value.

In this paper we consider the generalized random graph model $G(\mathbf{d})$ defined in [9]. For a sequence $\mathbf{d} = (d_1, \dots, d_n)$ the graph $G(\mathbf{d}) = (V, E)$ is constructed by letting two vertices $v_i, v_j \in V$ be connected with probability $p_{i,j} = d_i d_j / \sum_{k=1}^{n} d_k$. Furthermore, we assume that the degrees of the vertices follow a power law distribution, i.e., the number of vertices with expected degree i is proportional to $i^{-\beta}$ for some constant $\beta > 2$. We also assume that $d_{\min} = O(1)$ and $d_{\max} = \Theta(n^{1/\beta})$.

For a subset S of vertices, the volume $Vol(S)$ is defined as the sum of the expected degrees in S, i.e., $Vol(S) = \sum_{v \in S} d_v$. In particular, we have $Vol(G(\mathbf{d})) = \sum_{k=1}^{n} d_k$, and denote $\alpha = 1/Vol(G)$. Note that the classical random graph G_p can be viewed as a special case of $G(\mathbf{d})$ by taking $\mathbf{d} = (pn, \dots, pn)$ with $Vol(G_p) = pn^2$.

For a graph $G(\mathbf{d})$, let $A(\mathbf{d}) \in \{0,1\}^{n \times n}$ denote its *adjacency matrix.* Since $G(\mathbf{d})$ is undirected, $A(\mathbf{d})$ is symmetric. Column/row i of $A(\mathbf{d})$ contains 1's at the positions of all neighbors of v_i. In this paper, we mainly consider the Laplacian $L(\mathbf{d}) \in \mathbb{R}^{n \times n}$ of $G(\mathbf{d})$ defined as $L(\mathbf{d}) = D - A(\mathbf{d})$ with D containing the node degrees as diagonal entries and 0 elsewhere.

It is known that the eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$ of the Laplacian $L(\mathbf{d})$ are all nonnegative and 0 is a single eigenvalue iff the corresponding graph $G(\mathbf{d})$ is connected (e.g. [13]). In our proofs, we will often use the following lemma.

Lemma 1. *Let A be an $n \times n$ symmetric matrix. For any positive constants $c_1, \dots, c_n$ the largest eigenvalue $\lambda_n(A)$ satisfies the inequality*

$$\lambda_n(A) \leq \max_{1 \leq i \leq n} \frac{\sum_{j=1}^{n} c_j |a_{ij}|}{c_i}.$$

Proof. Let C be the diagonal matrix with c_i as the i^{th} diagonal entry. Both, A and $C^{-1}AC$ have the same eigenvalues, since for any $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$ such that $C^{-1}ACx = \lambda x$ we have $ACx = \lambda Cx$, and vice-versa. Then, using the Rayleigh principal we obtain

$$\begin{aligned}\lambda_n(A) = \max_{x \in \mathbb{R}^n} \frac{x^T C^{-1} A C x}{x^T x} &\leq \max_{x \in \mathbb{R}^n} \frac{x_+^T C^{-1} A_+ C x_+}{x^T x} \\ &\leq \max_{x \in \mathbb{R}^n} \frac{x^T C^{-1} A_+ C x}{x^T x} = \lambda_n(C^{-1} A_+ C),\end{aligned}$$

where x^T represents the transpose of x, x_+ describes the vector $(|x_1|, \ldots, |x_n|)$, and $(A_+)_{i,j} = |(A)_{i,j}|$ for any i, j. Due to the Perron-Frobenius Theorem, $\lambda_n(C^{-1}A_+C)$ is bounded from above by the maximum row sum of $C^{-1}A_+C$ (e.g. [30]), and the lemma follows. □

In order to show our main results, we also use the following Interlacing theorems (e.g. [22]).

Theorem 1. *Let A be a real symmetric $n \times n$ matrix and let A_i be the submatrix obtained from A by deleting its i^{th} row and column. Then, for any $j \in \{1, \ldots, n-1\}$ we have*

$$\lambda_j(A) \leq \lambda_j(A_i) \leq \lambda_{j+1}(A),$$

where $\lambda_j(A)$ denotes the jth smallest eigenvalue of A.

Theorem 2. *Let G be an undirected graph and let G' be the subgraph obtained from G by deleting one of its edges. Then, for any $j \in \{2, \ldots, n\}$ we have*

$$\lambda_{j-1} \leq \lambda'_j \leq \lambda_j,$$

where λ_j and λ'_j denote the jth smallest Laplacian eigenvalue of G and G', respectively.

In the following lemma we state an eigenvalue interlacing result for graphs containing an induced star as a subgraph.

Lemma 2. *Let $G = (V, E)$ be an undirected unweighted graph and let $G' = (V', E')$ be the graph defined by the set of nodes $V' = V \cup \{u_1, \ldots, u_k\}$ and set of edges $E' = E \cup \{\{v, u_1\}, \{v, u_2\}, \ldots, \{v, u_k\}\}$, where v is some vertex of V. Let λ_i and λ'_i denote the Laplacian eigenvalues of G and G', respectively. Then,*

$$\lambda'_{n'-i-1} \leq \lambda_{n-i} \leq \lambda'_{n'-i}$$

for any $i \geq 0$ with $\lambda_{n-i} > 1$, where n and n' represent the size of the graphs G and G', respectively.

Due to space limitations, we omit the proof of this lemma here.

3 Main Results

In this section, we first focus on the random power law graph model of [9] as described at the beginning of the previous section. Then, we consider the dynamical power law graph models of [2,3], and discuss the applicability of our methods in these graphs. In the next lemma we state lower bounds on the largest $\Theta(n^{1/\beta})$ Laplacian eigenvalues of $G(\mathbf{d})$.

Lemma 3. *Let $G(\mathbf{d})$ be the graph defined at the beginning of Section 2, and let $L(\mathbf{d})$ be its Laplacian with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$. Then, $\lambda_{n-j} \geq d_{n-j}(1 - o(1))$ for any $j = O(n^{1/\beta})$, with high probability*[1].

Proof. In order to show the lemma, we only consider the vertices with expected degree larger than $\delta n^{1/\beta}$ for some constant $\delta < 1$, their neighbors with expected degree less than $\log n$, and the edges between these two sets of nodes. Let the set of nodes with expected degree larger than $\delta n^{1/\beta}$ be denoted by U, and the set of their neighbors in $G'(\mathbf{d})$ be U'. Furthermore, let $E' = E \cap \{(v, v') \mid v \in U,\ v' \in U'\}$, and let $G'(\mathbf{d}) = (U \cup U', E')$.

Now we are going to show that a node with expected degree d for some $d > \delta n^{1/\beta}$ has $d(1 - o(1))$ neighbors in $G'(\mathbf{d})$, with probability $1 - o(1/n^2)$. Let v be a node with expected degree d. Due to the definition of $G(\mathbf{d})$, it holds that

$$Vol(V \setminus U') \leq \sum_{i=\lfloor \log n \rfloor}^{\infty} \Theta\left(\frac{n}{i^\beta} i\right) = O\left(\frac{n}{i^{\beta-2}}\right).$$

Since v is connected to some node $v_j \in V \setminus U'$ in $G(\mathbf{d})$ with probability $dd_j\alpha$, independently, and $\alpha = \Theta(1/n)$, we can model the occurrence of an edge between v and v_j by the independent random variable X_j, which is 1 with probability $dd_j\alpha$ and 0 with probability $1 - dd_j\alpha$, independently of any other $X_{j'}$. If $X = \sum_{v_j \in V\setminus U'} X_j$, then applying the Chernoff bounds [7,23], we obtain

$$\Pr\left[X \geq \left(1 + \Theta\left(\frac{1}{\sqrt[4]{d}}\right)\right) \mathbb{E}\left[\, X \,\right]\right] \leq e^{-\frac{\Theta(\mathbb{E}[\, X \,]/\sqrt{d})}{3}} = o\left(\frac{1}{n^2}\right),$$

for any $d > \delta n^{1/\beta}$, since $\mathbb{E}\left[\, X \,\right] = \Theta(d/\log^{\beta-2} n)$. Therefore, v has $d(1 - o(1))$ neighbors in $G'(\mathbf{d})$, with probability $1 - o(1/n^2)$.

Next, we show that $d(1-o(1))$ neighbors of v in $G'(\mathbf{d})$ have no other neighbors in U, with probability $1 - o(1/n^2)$. Let v_j be now a neighbor of v in U'. Then, v_j is connected to a node $v_{j''} \in U$ with probability $d_j d_{j''}\alpha$, independently of any other edge, and v_j has no other neighbor in U with probability $1 - d_j \cdot Vol(U) \cdot \alpha(1 \pm o(1)))$, independently of other edges between U and $U' \setminus \{v_j\}$. Thus, we model the occurrence of additional edges between v_j and U by the random variable X_j, which is 1 with probability $d_j \cdot Vol(U) \cdot \alpha(1 \pm o(1))$ and 0

[1] When we write "with high probability" or "w.h.p.", then we mean with probability at least $1 - o(1/n)$.

with probability $1 - d_j \cdot Vol(U) \cdot \alpha(1 \pm o(1))$, independently of any other $X_{j'}$. Then the Chernoff bounds [7,23] lead us to

$$\Pr\left[X \geq (1 + \Theta(\log n))\, \mathbb{E}[\, X\,]\right] \leq e^{-\Theta(\mathbb{E}[\, X\,]\log n)} \leq o\left(\frac{1}{n^2}\right),$$

where $X = \sum_{v_j \in U'} X_j$ and $\mathbb{E}[\, X\,] \leq O(\log n)$.

Let now $G''(\mathbf{d})$ be the graph obtained from $G'(\mathbf{d})$ by deleting all the edges which connect the nodes of U with the nodes having more than one neighbor in U. Due to the arguments described above, every node with expected degree $d > \delta n^{1/\beta}$ has $d(1 - o(1))$ neighbors in $G''(\mathbf{d})$, w.h.p.

Due to the previous arguments, $G''(\mathbf{d})$ is the disjoint union of $|U|$ many stars, and for any $d > \delta n^{1/\beta}$ there is a star of size $d(1 - o(1))$ in $G''(\mathbf{d})$, w.h.p. Since the largest laplacian eigenvalue of a star equals its size [13], for any $d > \delta n^{1/\beta}$ there exists an eigenvalue $\lambda'' = d(1 - o(1))$ of the Laplacian $L''(\mathbf{d})$ of $G''(\mathbf{d})$, w.h.p. Now, Theorem 2 implies that the eigenvalues of $L''(\mathbf{d})$ are smaller than the eigenvalues of $L(\mathbf{d})$, and the lemma follows. □

In the following theorem we derive upper bounds on the largest eigenvalues of $L(\mathbf{d})$, and show that these eigenvalues follow a power law distribution.

Theorem 3. *Let* $\tau = \lceil d_{\max}(1 - \sum_{k=d_{\min}}^{\infty} k^{-\beta} e^{-k} / \sum_{k=d_{\min}}^{\infty} k^{-\beta})\rceil$. *Then, the largest* $\sum_{j=\tau}^{d_{\max}} n/j^{\beta} \cdot (1 + o(1))$ *eigenvalues of* $L(\mathbf{d})$ *follow a power law distribution with exponent* β*, w.h.p.*

Proof. We use Lemma 1 to derive an upper bound on the largest eigenvalue of $L(\mathbf{d})$. Let the vertices of $G(\mathbf{d})$ be ordered according to their expected degree as in $\mathbf{d}$. To apply Lemma 1, we consider the vector $\mathbf{c}$ with $c_i = d_i$ if $d_i > \sqrt{n^{1/\beta}}$, and $c_i = \sqrt{n^{1/\beta}}$ otherwise.

In order to show that $\frac{\sum_{j=1}^{n} c_j |l_{ij}|}{c_i} \leq d_{\max}(1 + o(1))$ for any i, where $l_{i,j}$ denotes the (i, j) entry of $L(\mathbf{d})$, we consider two cases. First, we assume that $d_i > \sqrt{n^{1/\beta}}$. As in the proof of Lemma 3, we can use the Chernoff bounds [7,23] to show that v_i has $o(d_i)$ neighbors among the nodes with expected degree at least $\sqrt{n^{1/\beta}}$, with probability $1 - o(1/n^2)$. Therefore, if we denote by U the set of these nodes (without v_i), then

$$\frac{\sum_{v_j \in U} c_j |l_{ij}|}{c_i} \leq \frac{n^{1/\beta}}{d_i} o(d_i) = o(n^{1/\beta})$$

for any i, w.h.p. On the other hand, if we denote by U' the set of the nodes with expected degree at most $\sqrt{n^{1/\beta}}$, then we obtain

$$\frac{\sum_{v_j \in U'} c_j |l_{ij}|}{c_i} \leq \frac{\sqrt{n^{1/\beta}}}{d_i} d_i (1 + o(1)) = \sqrt{n^{1/\beta}}(1 + o(1)),$$

with probability $1 - o(1/n^2)$. Thus,

$$\frac{\sum_{v_j \in V} c_j |l_{ij}|}{c_i} \leq o(n^{1/\beta}) + d_i.$$

If $d_i \leq \sqrt{n^{1/\beta}}$, then we can, again, use the Chernoff bounds [7,23] as in the proof of Lemma 3 to conclude that v_i has $o(d_i + \log^2 n)$ neighbors in the set U, with probability $1 - o(1/n^2)$. Therefore,

$$\frac{\sum_{v_j \in U} c_j |l_{ij}|}{c_i} \leq \frac{n^{1/\beta}}{\sqrt{n^{1/\beta}}} o(d_i + \log^2 n) = o(n^{1/\beta}).$$

On the other hand,

$$\frac{\sum_{v_j \in U'} c_j |l_{ij}|}{c_i} \leq \frac{\sqrt{n^{1/\beta}}}{\sqrt{n^{1/\beta}}} d_i = d_i$$

(here we also assume that $v_j \notin U'$). Hence,

$$\frac{\sum_{v_j \in V} c_j |l_{ij}|}{c_i} \leq o(n^{1/\beta}) + 2d_i,$$

with probability $1 - o(1/n^2)$, and the claim follows.

In order to derive an upper bound on the largest $\sum_{j=\tau}^{n^{1/\beta}} n/j^\beta \cdot (1 + o(1))$ eigenvalues, we first show that a node v with degree $d = \Theta(n^{1/\beta})$ has at least $d \frac{\sum_{k=1}^{\infty} k^{-\beta} e^{-k}}{\sum_{k=1}^{\infty} k^{-\beta}} (1 - o(1))$ neighbors of degree 1[2], with probability $1 - o(1/n^2)$.

To show this, let $U_{d'}$ be the set of nodes of expected degree d' for some $d' = O(\log n)$. Applying the same arguments as in the proof of Lemma 3, v is connected with $d \cdot Vol(U_{d'}) \cdot \alpha (1 \pm o(1))$ vertices of $U_{d'}$. Let v'_j be a neighbor of v in $U_{d'}$. Let us define $Q(v) = V \setminus (N(v) \cup \{v\})$. Then, v'_j has no neighbor outside $N(v)$, other than v, with probability

$$\Pi_{v_k \in Q(v)} (1 - d' d_k \alpha) = (1 - \alpha)^{d' \sum_{v_k \in Q(v)} d_k} (1 - O(\sqrt{\alpha})) = \frac{1}{e^{d'}} (1 - o(1)).$$

Since v'_j has no neighbors outside $N(v)$ (other than v), with probability $1/e^{d'} \cdot (1 - o(1))$ and independently of all other vertices of $N(v)$, we can use again the Chernoff bounds as in Lemma 3 to show that, with probability $1 - o(1/n^2)$, $|N(v) \cap U_{d'}|/e^{d'} \cdot (1 - o(1))$ of the nodes of $N(v) \cap U_{d'}$ do not have any neighbors outside $N(v)$, other than v.

Now we consider the edges between the nodes of $N(v)$. Using similar arguments as before, it can be shown that $|N(v) \cap U_{d'}|(1 - o(1))$ nodes with expected degree $d' = O(\log n)$ have no neighbor of degree $\omega(\log n)$ in $N(v)$. Since two nodes $v_q, v_r \in N(v)$ of smaller expected degree than $O(\log n)$ are connected with probability $d_q d_r \alpha$, and $|N(v)| = \Theta(n^{1/\beta})$, the nodes which have expected degree $O(\log n)$ in $N(v)$ form a graph with less edges than a classical random graph $G_{O(\log^2 n/n)}$ of size $\Theta(n^{1/\beta})$. Applying the results of [4], we conclude that $|N(v) \cap U_{d'}|/e^{d'} \cdot (1 - o(1))$ of the nodes of expected degree d' in $N(v) \cap U_{d'}$ have no other neighbors than v. This implies that v has $d \frac{\sum_{k=d_{\min}}^{\infty} k^{-\beta} e^{-k}}{\sum_{k=d_{\min}}^{\infty} k^{-\beta}} (1 - o(1))$ neighbors of *actual* degree 1 (in $G(\mathbf{d})$), with probability $1 - o(1/n^2)$.

[2] Degree 1 means here the actual degree of these nodes and not their expected degree as listed in the vector $\mathbf{d}$.

Now, v_n (i.e., the node with largest degree in $G(\mathbf{d})$) forms with its neighbors of degree 1 a star in $G(\mathbf{d})$, and we can therefore apply Lemma 2 together with the arguments of the previous paragraphs to show that $\lambda_{n-1} \leq d_{n-1}(1-o(1))$, with probability $1-o(1/n^2)$. Iterating this technique as long as $d_{n-j} > d_{\max}(1-\sum_{k=d_{\min}}^{\infty} k^{-\beta}e^{-k}/\sum_{k=d_{\min}}^{\infty} k^{-\beta})(1+o(1))$, we obtain the desired upper bounds.

Combining Lemma 3 with the results above, we obtain the theorem. □

In Theorem 3 we considered the largest Laplacian eigenvalues of $G(\mathbf{d})$. Now we focus on the distribution of all eigenvalues and determine how many Laplacian eigenvalues lie in the interval (i,∞) for some $i=\omega(1)$. First, we compute lower bounds on the number of eigenvalues larger than i.

Lemma 4. *Let $G(\mathbf{d})$ be the random power law graph with $\beta \in (2,3)$ as defined at the beginning of Section 2, and let $L(\mathbf{d})$ denote its Laplacian. If $\rho(i) = \sum_{k=i}^{\infty} n_k = \frac{n}{(\beta-1)\sum_{k=d_{\min}}^{\infty} k^{-\beta}} i^{1-\beta}(1 \pm o(1))$, where n_k denotes the number of nodes with* actual *degree k, then for any constant $\epsilon > 0$ the Laplacian $L(\mathbf{d})$ has at least $\rho(i)(1-o(1))$ eigenvalues larger than $i(1-\epsilon)$, w.h.p.*

Proof. For simplicity we assume that $i = \Omega(\log^2 n)$. Let d_j^a denote the actual degree of node v_j, and let $G' = (V', E')$ be the graph obtained from $G(\mathbf{d})$ by deleting $d_j^a - i$ incident edges for any node v_j with $d_j^a > i$. Then, the largest degree in G' is i, and there are $\rho(i)(1 \pm o(1))$ vertices of degree i in G'. Furthermore, we construct the graph $G'' = (V'', E'')$ from G' by deleting $d_j^a - \log i$ incident edges for any node v_j with $d_j^a \in (\log i, i)$, and by deleting all edges which connect two lower degree nodes or two nodes of degree $i(1-o(1))$. Due to the arguments from the proof of Lemma 2, we still have $\rho(i)(1-o(1))$ vertices of degree $i(1-o(1))$ in G'', w.h.p. In the following paragraphs, however, we assume for simplicity that there are exactly $\rho(i)$ vertices of degree $i(1-o(1))$. We are now going to show for any constant $\epsilon > 0$ that the Laplacian L'' of G'' has $\rho(i)(1-o(1))$ eigenvalues which are larger than $i(1-\epsilon)$, w.h.p.

Let $\lambda_1'', \ldots, \lambda_n''$ be the eigenvalues of L''. As in the proof of Theorem 3, we can show that with high probability $\lambda_n'' \leq i(1+o(1))$. Now, we know that $\sum_{k=1}^{n} \lambda_k'' = \text{Trace}(L'')$, and since $(\lambda_k'')^q$ is an eigenvalue of $(L'')^q$ for any $k, q \in \mathbb{R}$, it holds that $\sum_{k=1}^{n} (\lambda_k'')^q = \text{Trace}((L'')^q)$ for any q. Now we show that $\text{Trace}((L'')^q) \geq i^q \rho(i)(1-o(1))$ for any constant $q > \beta - 1$.

Denote the entries of $(L'')^q$ by $l_{i,j}''(q)$, and let the vertices be ordered such that the first $\rho(i)$ vertices have degree i. To show the claim, we prove by induction on q that for any $q = O(1)$ it holds that $l_{i,i}''(q) = i^q(1 \pm o(1))$ if $i \leq \rho(i)$, and $|l_{i,i}''(q)| \leq \log^2 i \cdot (2i)^{q-2}(1+o(1))$ if $i > \rho(i)$. Within this induction we also show that in each row/column of $(L'')^q$ the absolute values of the non-diagonal entries $l_{i,j}''(q)$ are upper bounded by $(2i)^{q-1}(1 \pm o(1))$.

Obviously, the assumption holds for $q = 1$. It also holds for $q = 2$, since G'' is bipartite, and therefore $l_{i,i}''(2) = i^2(1 \pm o(1))$ for $i \leq \rho(i)$, $l_{i,i}''(2) = \log^2 i(1 \pm o(1))$ for $i > \rho(i)$, and $|l_{i,j}''(2)| \leq i(1+o(1))$ for $i \neq j$. For $q > 2$ we get

$$l_{i,i}''(q) = \sum_{k=1}^{n} l_{i,k}'' \cdot l_{k,i}''(q-1) = i^q(1-o(1)) \pm O(i \cdot (2i)^{q-2}),$$

if $i \leq \rho(i)$, and

$$|l''_{i,i}(q)| \leq \log i \cdot (2i)^{q-2}(1+o(1)) + \log i \cdot (2i)^{q-3}(1+o(1)),$$

if $i > \rho(i)$. Moreover,

$$|l''_{i,j}(q)| \leq \sum_{k=1}^{n} |l''_{i,k}| \cdot |l''_{k,j}(q-1)| \leq i \cdot (2i)^{q-2}(1+o(1)) + i \cdot (2i)^{q-2}(1+o(1)),$$

for any $i \neq j$, and the claim follows.

Since we assumed that q is bounded by some large constant, it holds that $l''_{i,i}(q) = i^q(1 \pm o(1))$ for $i \leq \rho(i)$, and $|l''_{i,i}(q)| = O(\log^2 i \cdot i^{q-2})$ otherwise. Since $\beta < 3$, for any $q > \beta - 1$ it holds that $\mathrm{Trace}((L'')^q) \geq i^q \rho(i)(1-o(1))$.

Now the proof can be completed by the method of Wigner [29]. We finish our proof, however, by only using some simple arguments. Let us assume that there exist the constants $\epsilon < 1$ and $\delta < 1$ such that $n - \rho(i)(1-\epsilon)$ eigenvalues are smaller than $\delta \cdot i$, and let $e \leq 1$ be a constant such that $\sum_{k=1}^{n-\rho(i)(1-\epsilon)} (\lambda''_k)^q = e \cdot i^q \rho(i)(1+o(1))$. Then,

$$\sum_{k=1}^{n-\rho(i)(1-\epsilon)} (\lambda''_k)^{q+s}(1+o(1)) \leq \delta^s e \cdot i^{q+s}\rho(i)(1+o(1)).$$

Since there exists an s such that $\delta^s e < \epsilon$, we get

$$\sum_{k=1}^{n} (\lambda''_k)^{q+s} \leq \delta^s e \cdot i^{q+s}\rho(i)(1+o(1)) + (1-\epsilon)i^{q+s}\rho(i)(1+o(1)) \leq (1-\epsilon')i^{q+s}\rho(i),$$

where $\epsilon' > 0$ is a constant, and the lemma follows. □

In this proof, we assumed that $i = \Omega(\log^2 n)$. However, using a more sophisticated analysis, the same result can also be shown for any $i = \omega(1)$. We omit the details due to space limitations.

In the next theorem we determine upper bounds on the number of Laplacian eigenvalues in the range (i, ∞) for some $i = \omega(1)$.

Lemma 5. *Let $G(\mathbf{d})$ be the random power law graph as defined at the beginning of Section 2, and let $L(\mathbf{d})$ denote its Laplacian. If $\rho(i) = \sum_{k=i}^{\infty} n_k = \frac{n}{(\beta-1)\sum_{k=d_{\min}}^{\infty} k^{-\beta}} i^{1-\beta}(1 \pm o(1))$, where n_k denotes the number of nodes with* actual *degree k, then for any constant $\epsilon > 0$ the Laplacian $L(\mathbf{d})$ has less than $\rho(i)$ eigenvalues which are larger than $i(1+\epsilon)$, w.h.p.*

Proof. As in the proof of Lemma 4, we modify the graph $G(\mathbf{d})$ in order to develop the desired upper bounds. However, now we add edges to the graph instead of deleting some. Again, we assume for simplicity that $i = \Omega(\log^2 n)$. Let the set U consist of the nodes v_j of degree $d_j > i$, and let $U' = V \setminus U$. As in the proof before, we can argue that any node $v_j \in U'$ with $d_j \leq i(1-o(1))$ has $o(i)$ neighbors in U, w.h.p.

Let $v_q \in U'$ be the vertex with maximum number of neighbors in U, and let d'_q represent the number of edges between v_q and U. Now we construct the graph $G' = (V', E')$ by connecting each vertex j of U' with $d'_q - d_j^{a,U}$ additional vertices from U, where $d_j^{a,U}$ denotes the number of edges between j and U in $G(\mathbf{d})$. Let L' be the Laplacian of G'. Then, due to Theorem 2 the eigenvalues of L' are larger than the eigenvalues of L.

Now let $L'_{\rho(i)}$ be the matrix obtained from L' by deleting all rows and columns corresponding to the $\rho(i)$ vertices (with degree larger than i) of U. Then, Theorem 1 implies that $\mu'_{n-\rho(i)} \geq \lambda'_{n-\rho(i)}$, where $\mu'_{n-\rho(i)}$ is the largest eigenvalue of $L'_{\rho(i)}$. In order to complete our proof, we need to show that $\mu'_{\rho(i)} \leq i(1+o(1))$, w.h.p. This would lead to $i(1+o(1)) \geq \lambda'_{n-\rho(i)} \geq \lambda_{n-\rho(i)}$, which implies that there are less than $\rho(i)$ eigenvalues which are larger than $(1+\epsilon)i$ for any constant $\epsilon > 0$, w.h.p.

In order to show that $\mu'_{\rho(i)} \leq i(1+o(1))$, we consider the graph G'' constructed from G' by deleting all vertices of U together with all their incident edges. Since every node of U' is connected to exactly d'_q nodes of U in G', the Laplacian L'' of G'' has the form $L'' = L'_{\rho(i)} - d'_q I$, where I is the identity matrix of the corresponding size. Using the techniques of Theorem 3 we conclude that the largest eigenvalue $\lambda''_{n-\rho(i)}$ of L'' is smaller than $i(1+o(1))$, w.h.p. Since $d'_q = o(i)$ (w.h.p.), the lemma follows. □

Combining Lemma 4 and 5, we obtain the following theorem.

Theorem 4. *Let $G(\mathbf{d})$ be the random power law graph with exponent $\beta \in (2,3)$ as defined at the beginning of Section 2, and let $L(\mathbf{d})$ denote its Laplacian. If $\rho(i) = \sum_{k=i}^{\infty} n_k = \frac{n}{(\beta-1)\sum_{k=d_{\min}}^{\infty} k^{-\beta}} i^{1-\beta}(1 \pm o(1))$, where n_k denotes the number of nodes with* actual *degree k, then $L(\mathbf{d})$ has $\rho(i)(1 \pm o(1))$ eigenvalues larger than i, w.h.p., for any $i = \omega(1)$.*

Theorem 4 implies that if $n(i)$ denotes the number of eigenvalues larger than i, for $i = \omega(1)$, then $n(i) \sim i^{1-\beta}$, w.h.p. Moreover, it follows from Theorem 4 that if $i = \omega(1)$, then there are $(\rho(i) - \rho((1+\epsilon)i))(1 \pm o(1))$ laplacian eigenvalues in any interval $(i, (1+\epsilon)i)$, w.h.p., where ϵ can be any constant. This implies that the distribution of the large eigenvalues is close to power law with exponent β.

The results of Theorem 3 and 4 can be generalized in different ways. Let us now consider the dynamical power law graph model of [3], or the more general growth-deletion model of [2]. In both cases, the graph G_t is built dynamically in the following way: We start with some graph G_0, and then in every step $t > 0$ we attach with some probability $p > 0$ a new vertex to the current graph G_{t-1}, and connect this vertex to m other vertices selected with probabilities proportional to their degrees, where m is some constant. In [3] $p = 1$, in [2] p can vary, whereby with probability $1-p$ some other actions are taken, like connecting two existing vertices, or deleting edges or vertices from the graph. In all these models, if $m = 1$, then we can use the fact that, with high probability, most high degree vertices are adjacent with many vertices of degree 1. Therefore, similar results to Theorem 3 can be proven for these graphs, too. However, there can be a large discrepancy between the node with the largest degree and the node with second largest degree, which makes it difficult to guarantee the results of Theorem 3 for $\Theta(n^{1/\beta})$ nodes. The results of Theorem 4, however, can be guaranteed with the same probability for the model of [2], whenever $\beta < 3$.

4 Conclusion

In this paper, we investigated the influence of the power law structure on the Laplacian spectrum of graphs. First, we computed the largest $\sum_{j=\tau}^{d_{\max}} \frac{n}{j^\beta} \cdot (1+o(1))$ eigenvalues of the Laplacian of the random power law graph $G(\mathbf{d})$, where $\tau = d_{\max}\left(1 - \sum_{k=d_{\min}}^{\infty} \frac{k^{-\beta}e^{-k}}{\sum_{k=d_{\min}}^{\infty} k^{-\beta}}\right)$, β is the exponent in the power law distribution of the degrees, and $d_{\min}$ and $d_{\max}$ denote the smallest and largest degree in $G(\mathbf{d})$, respectively. Then, we considered the distribution of all eigenvalues larger than $\omega(1)$, and showed that it is close to a power law with the same exponent β. We also considered some dynamically constructed power law graphs, and discussed the applicability of our methods in these graphs.

As we mentioned in the introduction, most real world network possess, apart from the power law degree distribution, some other properties (e.g. they have a large clustering coefficient), which are not present in the graphs studied here. Therefore, it would be interesting to analyze the behavior of the eigenvalues in such graphs. Nevertheless, we hope that our results might provide insights at a more general level, too.

References

1. D. Achlioptas and F. McSherry. Fast computation of low-ranked matrix approximation. In *Proc. of STOC'01*, pages 611–618, 2001.
2. W. Aiello, F.R.K. Chung, and L. Lu. Random evolution in massive graphs. In *Proc. of FOCS*, pages 510–519, 2001.
3. A. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286, 1999.
4. B. Bollobás. *Random Graphs.* Academic Press, 1985.
5. B. Bollobás, O. Riordan, J. Spencer, and G. Tusnády. The degree sequence of a scale-free random graph process. *Random Structures and Algorithms*, 18:279–290, 2001.
6. D. Callaway, J. Hopcroft, J. Kleinberg, M. Newman, and S. Strogatz. Are randomly grown graphs really random? *Physical Review E*, 64, 051902, 2001.
7. H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Stat.*, 23:493–507, 1952.
8. F.R.K. Chung. *Spectral Graph Theory*, volume 92 of *CBMS Regional conference series in mathematics.* American Mathematical Society, 1997.
9. F.R.K. Chung and L. Lu. Connected components in random graphs with given expected degre sequences. *Annals of Combinatorics*, 6:125–145, 2002.
10. F.R.K. Chung and L. Lu. The average distances in random graphs with given expected degrees. *Internet Mathematics*, 1:91–114, 2003.

11. F.R.K. Chung, L. Lu, and V. Vu. Eigenvalues of random power law graphs. *Annals of Combinatorics*, 7:21–33, 2003.
12. F.R.K. Chung, L. Lu, and V. Vu. The spectra of random graphs with given expected degrees. *Proceedings of National Academy of Sciences*, 100:6313–6318, 2003.
13. D.M. Cvetkovic, M. Doob, and H. Sachs. *Spectra of Graphs.* Johann Ambrosius Barth, 3rd edition, 1995.
14. R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.
15. R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems*, 35:305–320, 2002.
16. P. Erdős and A. Rényi. On random graphs I. *Publ. Math. Debrecen 6*, pages 290–297, 1959.
17. P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci. 5*, pages 17–61, 1960.
18. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *Comput. Commun. Rev.*, 29:251–263, 1999.
19. C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral partitioning with indefinite kernels using the nyström extension. In *European Conference on Computer Vision*, 2002.
20. E.N. Gilbert. Random graphs. *Ann. Math. Statist. 30*, pages 1141–1144, 1959.
21. C. Godsil and G. Royle. *Algebraic Grapg Theory.* Springer Verlag, 2001.
22. W. Haemers. Interlacing eigenvalues and graphs. *Linear Algebra Appl.*, 227/228:593–616, 1995.
23. T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 36(6):305–308, 1990.
24. H. Jeong, B. Tomber, R. Albert, Z. Oltvai, and A.L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407:378–382, 2000.
25. J. Kleinberg. Authoritive sources in a hyperlinked environment. In *Proc. of SODA'99*, 1999.
26. J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagoplan, and A. Tomkins. The Web as a graph: Measurements, models, and methods. In *Proc. of COCOON'99*, pages 1–17, 1999.
27. M. Mihail and C. Papadimitriou. On the eigenvalue power law. In *Proc. of RANDOM'02*, pages 254–262, 2002.
28. M. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
29. E.P. Wigner. On the distribution of the roots of certain symmetric matrices. *The Annals of Mathematics*, 67:325–327, 1958.
30. J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford University Press, 1965.

Multicast Transmissions in Non-cooperative Networks with a Limited Number of Selfish Moves

Angelo Fanelli, Michele Flammini, Giovanna Melideo, and Luca Moscardelli

Dipartimento di Informatica
Università di L'Aquila
Via Vetoio, Coppito 67100 L'Aquila
{angelo.fanelli, flammini, melideo, moscardelli}@di.univaq.it

Abstract. We study a multicast game in communication networks in which a source sends the same message or service to a set of destinations and the cost of the used links is divided among the receivers according to given cost sharing methods. Assuming a selfish and rational behavior, each receiving user is willing to select a strategy yielding the minimum shared cost. A Nash equilibrium is a solution in which no user can decrease its payment by adopting a different strategy, and the price of anarchy is defined as the worst case ratio between the overall communication cost yielded by an equilibrium and the minimum possible one. Nash equilibria requiring an excessive number of steps to be reached or being hard to compute or not existing at all, we are interested in the determination of the price of anarchy reached in a limited number of rounds, each of which containing at least one move per receiving user. We consider different reasonable cost sharing methods, including the well-known Shapley and egalitarian ones, and investigate their performances versus two possible global criteria: the overall cost of the used links and the maximum shared cost of users. We show that, even in case of two receivers making the best possible move at each step, the number of steps needed to reach a Nash equilibrium can be arbitrarily large. Moreover, we determine the cost sharing methods for which a single round is already sufficient to get a price of anarchy comparable to the one at equilibria, and the ones not satisfying such a property. Finally, we show that finding the sequence of moves leading to the best possible global performance after one-round is already an intractable problem, i.e., NP-hard.

Keywords: Multicast, Nash equilibria, price of anarchy, limited number of best responses.

1 Introduction

Multicast protocols like the IP over the Internet are bandwidth-conserving technologies that reduce traffic over a network by simultaneously delivering the same message or service of a given source station s to a set R of receivers. Applications that take advantage of multicast include videoconferencing, corporate communications, distance learning, distribution of software, stock quotes, and news [8]. Unlike unicast transmissions, where in order to send the same message to multiple receivers a source has to send a copy of this message to each receiver, in multicast transmissions the source sends a single message to all the receivers and when the message reaches a branch point the router duplicates it and then sends a copy over each down-stream link. As the bandwidth used by a transmission is not attributable to a single receiver, a natural arising issue is that of finding a way to distribute the cost among all the receivers in some fashion.

In large-scale scenarios, such as the Internet, there is no authority possible to enforce a centralized traffic management. In such situations, game theory and especially the concepts of Nash equilibria [24] are a suitable framework. If we allow as strategies for each receiver $t \in R$ the set $\mathcal{P}_t$ of the paths from s to t (briefly, (s,t)-paths), a solution is obtained as the outcome of a $|R|$-player game in which receivers t (players) can sequentially modify their strategy by selfishly choosing a different (s,t)-path with the aim of minimizing their shared cost, expressed in terms of a publicly known *cost sharing method*, which specifies how to share the overall cost of the transmission among the receivers belonging to R. Namely, a solution is a *path system* $\mathcal{P}$ containing an (s,t)-paths for every receiver $t \in R$, and the global cost $cost(\mathcal{P})$ to be shared among all the receivers according to a cost sharing method is obtained by summing up the cost of all the links belonging to $\mathcal{P}$. A path system $\mathcal{P}$ is a Nash equilibrium if no player has an incentive to secede in favor of a different solution.

The main algorithmic issues coming from this model include: proving the existence of a Nash equilibrium[1], proving the convergence to a Nash equilibrium from any initial configuration of the players' strategies, estimating the convergence time (i.e. the number of moves necessary to reach an equilibrium starting from an arbitrary configuration), finding Nash equilibria having particular properties (for instance, the one minimizing the global cost or minimize the maximum shared cost), and measuring the *price of anarchy* [20], that is the worst case ratio between the optimal social solution and a Nash equilibrium. Often Nash equilibria may not exist or it may be hard to compute them or the time for convergence to Nash equilibria may be extremely long, even if the players always choose a *best response move*, i.e. a move providing them the smallest possible shared cost. Thus, recent research effort [23] concentrated in the evaluation of the speed of convergence (or non-convergence) to an equilibrium in terms of *covering walks*, where a covering walk consists of a sequence of best response moves of the receivers, with each receiver appearing at least once in each walk. As a special case, a *one-round walk* is defined as a covering walk such that each receiver plays exactly one best response move. Moreover, another important issue is evaluating the loss of social performance in selfish evolutions with a (polynomially) bounded number of moves, not necessary terminating in a Nash equilibrium.

[1] Indeed, Nash proved that a randomized equilibrium always exists, while we are interested in pure Nash equilibria.

Related Work. Several games [11,12,15,22,28,31] have been shown to possess pure Nash equilibria or to converge to a pure Nash equilibrium independently from their starting state. An interesting work estimating the convergence time to Nash equilibria is [10] and in [7] finding Nash equilibria having particular properties has been shown to be NP-complete. Considerable research effort has been also devoted to analyze the price of anarchy in different settings, such as in wireless and all-optical networks [3,5,19,21,29].

The multicast cost sharing problem has been largely investigated both in standard networks [1,13,14,18,25,27] and in wireless networks [4,26], where the cost shared among the receivers is the overall power consumption, also in the *mechanism design* framework.

Mirrokni and Vetta [23] addressed the convergence to approximate solutions in basic-utility and valid-utility games. They proved that starting from any state, one-round of selfish behavior of players converges to a 1/3-approximate solution in basic-utility games. Goemans, Mirrokni and Vetta [17] studied a new equilibrium concept (i.e. sink equilibria) inspired from convergence on best-response walks and proved a fast convergence to approximate solutions on best response walks in (weighted) congestion games. Other related papers studied the convergence for different classes of games such as load balancing games [10], market sharing games [16], and potential and cut games [6].

Our Contribution. In this paper we consider the multicast games induced by four natural cost sharing methods which distribute the cost as follows: (i) in an *egalitarian* way, that is by equally distributing the overall cost among all the receivers; (ii) in a *path-proportional* way, that is by distributing the cost of each link among its down-streaming receivers proportionally to the overall cost their chosen path requires; (iii) in an *egalitarian-path-proportional* way, that is by distributing the overall transmission cost among all the receivers proportionally to the cost of their chosen path; (iv) by applying the definition of the *Shapley Value* [30], that is by equally distributing the cost of each link among all the down-stream receivers.

We first prove that, while the game yielded by the path-proportional cost sharing method in general does not admit a Nash equilibrium, the other three methods yield games that always converge to a Nash equilibrium starting from any initial configuration. We then show that the price of anarchy for the game yielded by the egalitarian method is unbounded and provide matching upper and lower bounds for the price of anarchy of the other two convergent games with respect to two different social cost functions, that is the overall transmission cost (function *cost*), which coincides with the sum of all the shared costs, and the maximum shared cost paid by the receivers (function *max*). Unfortunately, for both such metrics the price of anarchy is the worst possible one, that is equal to the number of receivers. Moreover, for the methods inducing convergent games, we prove that even with only two receivers, the number of best responses needed to reach a Nash equilibrium starting from an arbitrary configuration can be arbitrarily large.

Motivated by the previous results, we evaluate the price of the anarchy after a limited number of best responses, for all the proposed methods including the path-proportional one, which may do not admit a Nash equilibrium. We prove that for the egalitarian and path-proportional methods the price of the anarchy is unbounded for any sequence of best response moves and one-round walks, respectively. For the more interesting egalitarian-path-proportional and Shapley value methods we provide tight or almost tight bounds for one-round and covering walks. Such results have been determined for both the two different global cost functions *cost* and *max*.

Finally, we show that finding the best permutation of receivers moves leading to the lowest possible social cost after a one-round walk is already an intractable problem, i.e., NP-hard.

The paper is organized as follows. In the next section we present some basic definitions and notation. In Section 3 we present some preliminaries results concerning the Nash equilibria for the multicast routing problem, and we show that the number of moves necessary to reach a Nash equilibrium can be arbitrarily high. In Section 4 we provide upper and lower bounds on the price of anarchy after a one-round walk or a covering walk, for the social function *cost*. In Section 5 we prove the intractability result and in Section 6 we briefly extend the proofs to the social function *max*. Finally, in Section 7 we give some conclusive remarks and discuss some open questions.

Due to space limitation, many proofs are only sketched or omitted. More details will appear in the full version of the paper.

2 Definitions and Notation

A communication network is usually modelled as a graph $G(V, E, c)$ in which $V = \{1, \ldots, n\}$ is a set of intercommunicating nodes, $E \subseteq V \times V$ is a set of m links between the nodes and $c : E \mapsto I\!R^+$ is a *cost function* associating to each link (t, t') a *transmission cost*, that is the cost for exchanging messages between nodes t and t'. Given a distinguished source node $s \in V$, we identify with $R \subseteq V - \{s\}$ the set of all the nodes interested in receiving the transmission from the source s. Let us denote by $c(p_t) = \sum_{e \in p_t} c(e)$ the overall transmission cost of a path p_t.

Given a path system $\mathcal{P}$, the global cost to be shared among all the receivers is obtained by summing up the cost of all the links belonging to $\mathcal{P}$, i.e., $cost(\mathcal{P}) = \sum_{e \in E'} c(e)$, where $E' = \bigcup_{p \in \mathcal{P}} \bigcup_{e \in p} \{e\}$.

A *cost sharing method* is a function $\mathcal{M}$ which, given a set of receivers R with an associated path system $\mathcal{P}$, distributes among all the receivers the total cost $cost(\mathcal{P})$ associated with $\mathcal{P}$ in such a way that $\sum_{t \in R} \mathcal{M}(\mathcal{P}, t) = cost(\mathcal{P})$, where $\mathcal{M}(\mathcal{P}, t)$ is the cost attributed to the receiver t.

We consider the following four natural cost sharing methods:

– $\mathcal{M}_1$ (*egalitarian* [9]) distributes the cost by equally sharing the global cost among all the receivers, i.e., $\mathcal{M}_1(\mathcal{P}, t) = \frac{cost(\mathcal{P})}{|R|}$.

– $\mathcal{M}_2$ (*path-proportional*) distributes the cost of each link $e \in E'$ among all the down-streaming receivers t' using e proportionally to the overall cost their chosen (s, t')-path requires, i.e., $\mathcal{M}_2(\mathcal{P}, t) = \sum_{e \in p_t} c(e) \frac{c(p_t)}{\sum_{t' : e \in p_{t'}} c(p_{t'})}$.
– $\mathcal{M}_3$ (*egalitarian-path-proportional*) distributes the overall cost among all the receivers proportionally to the cost of their chosen path, i.e., $\mathcal{M}_3(\mathcal{P}, t) = cost(\mathcal{P}) \frac{c(p_t)}{\sum_{t' \in R} c(p_{t'})}$
– $\mathcal{M}_4$ (*Shapley* [30]) equally distributes the cost of each link among all the receivers using it, i.e., $\mathcal{M}_4(\mathcal{P}, t) = \sum_{e \in p_t} \frac{c(e)}{l(\mathcal{P}, e)}$ where $l(\mathcal{P}, e) = |\{t \in R \mid e \in p_t,\, p_t \in \mathcal{P}\}|$ is the number of receivers using link e for their transmission.

A *Nash equilibrium* for $\mathcal{G}$ is a path system $\mathcal{P}$ such that $\forall t \in R$ and path $p'_t \in \mathcal{P}_t$ inducing a new path system $\mathcal{P}' = \mathcal{P} \setminus \{p_t\} \cup \{p'_t\}$, it holds $\mathcal{M}(\mathcal{P}, t) \leq \mathcal{M}(\mathcal{P}', t)$. Denoting with $\mathcal{N}$ the set of all the possible Nash equilibria for the game $\mathcal{G}$, the *price of anarchy* is defined as the worst case ratio among the Nash versus optimal performance, that is $\rho(\mathcal{G}) = \max_{\mathcal{P} \in \mathcal{N}} \frac{cost(\mathcal{P})}{cost(\mathcal{P}^*)}$ where $\mathcal{P}^*$ is a path system of minimum cost for the multicast routing.

In order to model the selfish behavior of the receivers, let us introduce the notion of *state graph*.

Definition 1. *A* state graph *is a directed graph having a node for any possible path system $\mathcal{P}$ and an arc $(\mathcal{P}, \mathcal{P}')$ with label t if $\mathcal{P}$ and $\mathcal{P}'$ can differ only for the choice of t and both these conditions are met: (i) $\mathcal{M}(\mathcal{P}', t) \leq \mathcal{M}(\mathcal{P} - \{p_t\} \cup \{p'_t\}, t)$ for any $p'_t \in \mathcal{P}_t$; (ii) if $\mathcal{P} \neq \mathcal{P}'$, $\mathcal{M}(\mathcal{P}', t) < \mathcal{M}(\mathcal{P}, t)$.*

Notice that the graph may contain loops, and there is an arc $(\mathcal{P}, \mathcal{P}')$ labelled t if and only if t, starting from $\mathcal{P}$, can play a best response move such that the resulting path system is $\mathcal{P}'$. Given a best response walk starting from an arbitrary state, we are interested in the social value of the last state of the walk. With a little abuse of notation, we will call *price of anarchy of a walk* the worst case ratio between such social value and the social optimum. Notice that if we do not allow every player to make a best response on a walk P, then we cannot bound such a price since the actions of a single player may be very important for producing solutions of high social value. Motivated by this simple observation, Mirrokni and Vetta [23] introduced the following models that capture the intuitive notion of a fair sequence of moves:

One-round walk. Consider an arbitrary ordering of all receivers $i_1, \ldots, i_{|R|}$. A walk of length $|R|$ in the state graph is a one-round walk if its arcs are labelled $i_1, \ldots, i_{|R|}$ in this order.
Covering walk. A walk in the state graph is a covering walk if for each player i, it has at least one arc with label i.
k-Covering walk. A walk in the state graph is a k-covering walk if it can be split in k disjoint covering walks.

Note that unless otherwise stated, all walks are assumed to start from an arbitrary initial state.

3 Existence and Convergence to Nash Equilibria

As an extension of our work on Nash equilibria in multicast transmissions in wireless networks [2], it is not difficult to prove the following results on the existence of Nash equilibria and to provide matching upper and lower bounds for the price of anarchy.

Theorem 1. *The games $\mathcal{G} = (G, R, \mathcal{M}_i)$, $i \in \{1, 3, 4\}$, always converge to a Nash equilibrium, for any network G and receivers' set R.*

On the other hand, the game $\mathcal{G} = (G, R, \mathcal{M}_2)$ may not have a Nash equilibrium. Anyway, in the following we will evaluate the price of anarchy for the subset of instances admitting equilibria.

Theorem 2. *The price of anarchy of the multicast transmission game $\mathcal{G} = (G, R, \mathcal{M}_i)$ is unbounded for $i = 1$, is equal to $|R|$ for $i \in \{2, 4\}$ and is between $\frac{|R|+1}{2}$ and $|R|$ for $i = 3$.*

For the cost sharing methods inducing games which always admit a Nash equilibrium, the number of best response moves necessary to reach an equilibrium starting from an arbitrary configuration is unbounded even for a game with two receivers. In fact, the following theorem holds.

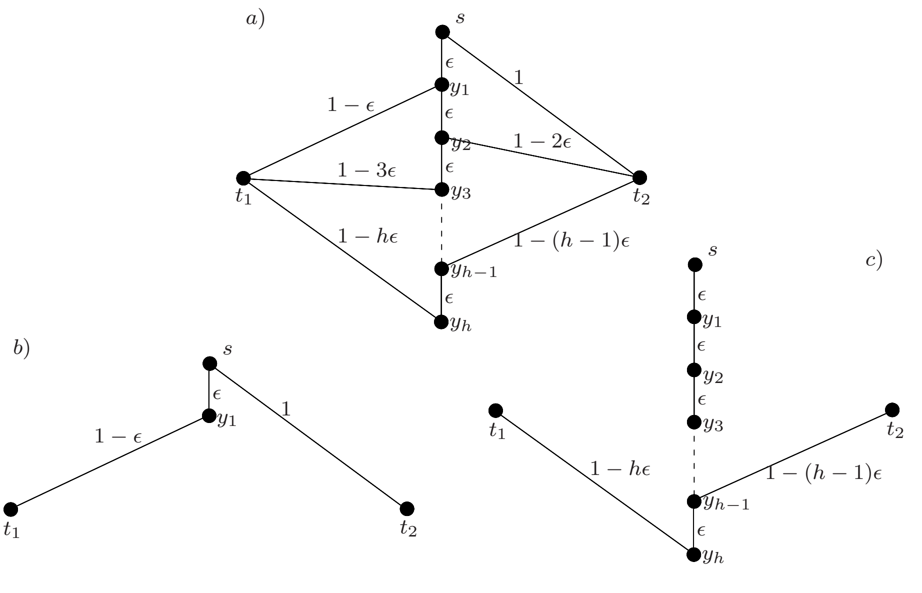


Fig. 1. a) The communication network. b) The initial configuration. c) The Nash equilibrium.

Theorem 3. *Given any integer $h > 0$, there exist a network G and an initial state starting from which the number of best response moves necessary to reach a Nash equilibrium for the game $\mathcal{G} = (G, R, \mathcal{M}_i)$, $i \in \{1, 3, 4\}$, is greater than h even if $|R| = 2$.*

Proof. Consider the communication network depicted in Figure 1a, where $R = \{t_1, t_2\}$ and $\epsilon > 0$ is such that $h\epsilon < \frac{1}{2}$. Consider the evolution of the game induced by the cost sharing method $\mathcal{M}_i$ for $i \in \{1, 3, 4\}$, starting from the state corresponding to the path system depicted in Figure 1b. Moreover assume that t_1 and t_2 move alternately, starting from t_1, and each receiver always chooses the path with the smallest number of links among those with the smallest cost.

Since $1 - h\epsilon > \frac{1}{2}$, no receiver can choose as its best move a path containing more than one edge of cost $1 - i\epsilon, i = 0, 1, \ldots, h$. It's easy to see that the number of moves executed in order to reach the equilibrium is h. More precisely, considering the ordered list of moves $j = 1, 2, \ldots, h$, the odd moves are executed by t_1 and the even ones by t_2; each receiver chooses the path containing the edge of cost $1 - j\epsilon$ and j edges of cost ϵ. □

4 One-Round and Covering Walks

In this section, we analyze the price of anarchy after a limited number of best response moves, with respect to the overall transmission cost social function. More precisely, we provide lower bounds for one-round walks and upper bounds for covering walks; since a one-round walk is a special case of a covering one, the lower and upper bounds also hold for covering and one-round walks, respectively.

For the cost sharing methods $\mathcal{M}_1$ and $\mathcal{M}_2$, it is possible to show that the price of anarchy after one-round walks is unbounded.

Theorem 4. *Given any integer h, there exists an instance of the game $\mathcal{G} = (G, R, \mathcal{M}_1)$ for which the price of anarchy after a one-round walk is at least h.*

Notice that this result holds for any sequence of best response moves, including k-covering paths for any $k \geq 1$.

Concerning the cost sharing method $\mathcal{M}_2$, as shown in Theorem 1, it may not admit a Nash equilibrium. However, again it is interesting to estimate the price of anarchy after a limited number of best response moves. Unfortunately, also in this case the price of anarchy after a one-round walk is unbounded.

Theorem 5. *Given any integer h, there exists an instance of the game $\mathcal{G} = (G, R, \mathcal{M}_2)$ for which the price of anarchy after a one-round walk is at least h.*

The remaining cost sharing methods $\mathcal{M}_4$ and $\mathcal{M}_3$ are more interesting since it is possible to bound the price of anarchy after a one-round or a covering walk in a tight or almost tight way, respectively.

4.1 Egalitarian Path Proportional Cost Sharing Method $\mathcal{M}_3$

For the cost sharing method $\mathcal{M}_3$ we provide matching upper and lower bounds for $|R| = 2$, and almost matching lower and upper bounds for $|R| > 2$.

Theorem 6. *Given any $\epsilon > 0$, there exists an instance of the game $\mathcal{G} = (G, R, \mathcal{M}_3)$ where $|R| = 2$ for which the price of anarchy after a one-round walk is at least $3 - \epsilon$.*

Theorem 7. *The price of anarchy after a covering walk for the game $\mathcal{G} = (G, R, \mathcal{M}_3)$ where $|R| = 2$ is at most 3.*

Starting from Theorem 2, it is not difficult to prove the following theorem.

Theorem 8. *There exists an instance of the game $\mathcal{G} = (G, R, \mathcal{M}_3)$ for which the price of anarchy after a one-round walk is at least $|R|$.*

In order to derive a suitable upper bound, in the following lemma we first show an interesting property correlating the cost of the path chosen by a receiver t doing a best response move with the one of the shortest $(s - t)$ path.

Lemma 1. *Let p_t be the path chosen by player t via a best response move, and m_t be the minimum cost path connecting s to t. Then, $c(p_t) \leq (1 + \phi)c(m_t)$, where $\phi \approx 1,618$ is the golden number.*

Proof. Let $A = \sum_{t' \in R - \{t\}} c(p_{t'})$ be the sum of the cost of the path used by the other receivers and B be the overall cost of the path system discarding t, just before the best response move of t. Moreover, let $x = c(p_t)$ and $m = c(m_t)$. Clearly, $x \geq m$. Since t plays a best response move, the following *base inequality* holds:

$$\frac{x}{A + x}(B + \delta_x) \leq \frac{m}{A + m}(B + \delta_m),$$

where δ_x and δ_m are the costs of the edges in x and m, respectively, not contained in the paths used by the other receivers.
We distinguish two different cases.

- $\delta_x \geq \frac{x}{1+\phi}$. Since by the base inequality it must hold that $B + \delta_x \leq B + \delta_m$, we obtain $B + \frac{x}{1+\phi} \leq B + \delta_x \leq B + \delta_m \leq B + m$. Thus, $x \leq (1 + \phi)m$.
- $\delta_x < \frac{x}{1+\phi}$. We have that $B \geq x - \frac{x}{1+\phi}$, which implies $x \leq \phi B$.
  The proof is again divided into two disjoint subcases.
  - If $A \cdot B \leq m^2$, since $A \geq B$, it follows that $B \leq m$. Thus, $x \leq \frac{1+\phi}{\phi} m < (1 + \phi)m$.
  - If $A \cdot B > m^2$, by the base inequality we obtain that $x \leq \frac{AB+Am}{AB-m^2} m = \frac{B+m}{B-\frac{m^2}{A}} m \leq \frac{B^2+Bm}{B^2-m^2} m$.
    Moreover, since $x \leq \phi B$, we have that $\frac{x}{m} \leq \min\{\frac{\phi B}{m}, \frac{B^2+Bm}{B^2-m^2}\}$.
    Since, fixed B, $\frac{\phi B}{m}$ is a decreasing function in m and $\frac{B^2+Bm}{B^2-m^2}$ is increasing in m, the minimum is maximized in the intersection of the two functions, which is obtained for $m = \frac{B}{\phi}$, where both of them assume value $1+\phi$. □

We are now ready to prove the following theorem, that provides an almost matching upper bound for the price of anarchy after a covering walk.

Theorem 9. *The price of anarchy after a covering walk for the game $\mathcal{G} = (G, R, \mathcal{M}_3)$ is at most $(1 + \phi)(|R| - 1) + 1$, where $\phi \approx 1,618$ is the golden number.*

4.2 Shapley Cost Sharing Method $\mathcal{M}_4$

For the cost sharing method $\mathcal{M}_4$ we provide matching upper and lower bounds. Unfortunately, the price of anarchy after a covering walk is very high, i.e. its order is quadratic in the number of receivers.

Theorem 10. *Given any $\epsilon > 0$, there exists an instance of the game $\mathcal{G} = (G, R, \mathcal{M}_4)$ for which the price of anarchy after a one-round walk is at least $\frac{|R|(|R|+1)}{2} - \epsilon$.*

Theorem 11. *The price of anarchy after a covering walk for the game $\mathcal{G} = (G, R, \mathcal{M}_4)$ is at most $\frac{|R|(|R|+1)}{2}$.*

Proof. We refer to the *last move* of a receiver as its last move in the covering walk. Let $t_1, \ldots, t_{|R|}$ be the sequence of the receivers such that $i < j$ if and only if the last move of t_i precedes the last move of t_j in the covering walk. Moreover, for $i \in \{1, \ldots, |R|\}$, let m_i, x_i and p_i be a shortest path connecting s to t_i, the cost payed by player t_i just after its last move and the s-t_i path chosen by t_i at its last move, respectively.

Consider the last move of receiver t_i. Clearly, $x_i \le c(m_i)$, otherwise m_i would be a better path for t_i. For each edge e of p_i, let $x_{i,e}$ and $y_{i,e}$ be the payment of t_i for edge e just after its last move and at the end of the covering walk, respectively, and let $h_{i,e} = h'_{i,e} + h''_{i,e}$ be the number of receivers sharing edge e with t_i just after its last move, where $h'_{i,e}$ is the number of receivers t_j with $j < i$, i.e. not moving after the last move of t_i.

The payment of t_i at the end of the covering walk is

$$\sum_{e \in p_i} y_{i,e} \le \sum_{e \in p_i} x_{i,e} \frac{1 + h'_{i,e} + h''_{i,e}}{1 + h'_{i,e}} \le \sum_{e \in p_i} x_{i,e}(1 + h''_{i,e}) \le$$

$$\le (|R| - i + 1) \sum_{e \in p_i} x_{i,e} \le (|R| - i + 1)c(m_i).$$

Since an optimal routing has cost at least equal to $max\{c(m_1), \ldots, c(m_{|R|})\}$, the price of anarchy after a covering walk is at most

$$\frac{|R|c(m_1) + (|R| - 1)c(m_2) + \ldots + c(m_{|R|})}{max\{c(m_1), \ldots, c(m_{|R|})\}} \le |R| + (|R| - 1) + \ldots + 1 = \frac{|R|(|R| + 1)}{2}.$$

□

5 Computing the Best One-Round Evolution is NP-Hard

A network provider could be interested in determining a proper permutation of the receivers such that, letting the receivers move in the specified order, the final configuration after a one-round walk is ensured to have the lowest possible social cost. Under this scenario, we now prove that determining such a permutation for the games $\mathcal{G} = (G, R, \mathcal{M}_i)$ for $i \in \{1, 2, 3, 4\}$ is computationally hard.

Theorem 12. *Consider the cost sharing methods $\mathcal{M}_i$, $i \in \{1,2,3,4\}$. Computing the permutation of receivers such that, letting the receivers move in the specified order, the final configuration after the one-round walk is ensured to have the lowest possible social cost is an NP-hard problem.*

Finally, by using the same reduction, since the best response moves are always unique, also the problem of determining the best one-round, i.e. the permutation and the best response move for each agent, leading to the best social value, is NP-Hard.

6 The Maximum Shared Cost Social Function

In this section we show how to extend our results to the case in which the considered social function *max* is given by the maximum shared cost of the receivers, that is $max(\mathcal{P}, \mathcal{M}_i) = max_{t \in R} \mathcal{M}_i(\mathcal{P}, t)$, for $i \in \{1,2,3,4\}$.

Due to space limitation, we just restrict in outlining the basic differences with respect to the previous analyzed social function given by the overall cost of the used path system. More details will appear in the full version of the paper.

Concerning the price of anarchy of Nash equilibria, it remains unbounded for the cost sharing method $\mathcal{M}_1$, and is again equal to $|R|$ for the cost sharing methods $\mathcal{M}_3$ and $\mathcal{M}_4$.

For the cost sharing methods $\mathcal{M}_1$ and $\mathcal{M}_2$, the price of anarchy remains unbounded for k-covering walks, with any k, and one-round walks, respectively.

For the cost sharing method $\mathcal{M}_3$, in networks with 2 receivers we can derive a lower bound equal to 4 for a one-round walk, and an exactly matching upper bound for a covering walk. Moreover, for more than 2 receivers we have a lower bound equal to $\frac{|R|+1}{2}$ for a one-round walk, and an upper bound of $(1+\phi)|R|$ for a covering walk, where $\phi \approx 1,618$ is the golden number.

For the cost sharing method $\mathcal{M}_4$, we can derive matching lower and upper bounds equal to $|R|^2$ for a one-round walk and a covering walk, respectively.

Finally, the problem of determining a proper permutation of the receivers such that, letting the receivers move in the specified order, the final configuration after a one-round walk is ensured to have the lowest possible social cost, remains *NP-hard* in the case of the social function *max*, for all the four considered cost sharing methods.

7 Conclusions

We have investigated the price of anarchy of the selfish game arising by multicasting in non-cooperative networks with respect to four basic cost sharing methods and two different social functions. Many question are left open.

First of all, for the Shapley cost sharing method $\mathcal{M}_4$, it would be nice to determine the minimum number of rounds sufficient to guarantee a price of anarchy proportional to the one at equilibrium. In fact, while $\mathcal{M}_3$ satisfies this property and for $\mathcal{M}_1$ the price of anarchy can be unbounded after any number of rounds, $\mathcal{M}_4$ is the only cost sharing method that after one-round has a price of anarchy quadratic with respect to the one at equilibrium. Even if we do not have a formal proof yet, for $\mathcal{M}_4$ we conjecture that two rounds are already enough.

On this respect, an exception holds for $\mathcal{M}_2$, for which we have proven that the price of anarchy is unbounded after one-round, but there may not exist an equilibrium.

It would also be nice to refine our results by determining when possible the number of rounds needed to exactly reach the price of anarchy at equilibrium.

Following the framework of [6], a possible extension is that of considering the game performances starting from an initial empty configuration in which no agent has selected any strategy.

Finally, a worth investigating issue is that of considering also the price of stability, that is the best possible performance achievable at equilibrium or after a fixed number of moves.

References

1. A. Archer, J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Approximation and collusion in multicast cost sharing. *Games and Economic Behavior*, 47(1):36–71, 2004.
2. V. Bilò, M. Flammini, G. Melideo, and L. Moscardelli. On nash equilibria for multicast transmissions in ad-hoc wireless networks. In *ISAAC*, volume 3341 of *Lecture Notes in Computer Science*, pages 172–183. Springer, 2004.
3. V. Bilò, M. Flammini, and L. Moscardelli. On nash equilibria in non-cooperative all-optical networks. In *STACS*, volume 3404 of *Lecture Notes in Computer Science*, pages 448–459. Springer, 2005.
4. V. Bilò, C. Di Francescomarino, M. Flammini, and G. Melideo. Sharing the cost of multicast transmissions in wireless networks. In *SPAA*, pages 180–187. ACM, 2004.
5. V. Bilò and L. Moscardelli. The price of anarchy in all-optical networks. In *SIROCCO*, volume 3104 of *Lecture Notes in Computer Science*, pages 13–22. Springer, 2004.
6. G. Christodoulou, V. S. Mirrokni, and A. Sidiropoulos. Convergence and approximation in potential games. In *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2006.
7. V. Conitzer and T. Sandholm. Complexity results about nash equilibria. In *IJCAI*, pages 765–771. Morgan Kaufmann, 2003.
8. S. Deering and D. CHeriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8:85–110, 1990.
9. B. Dutta and D. Ray. A concept of egalitarianism under participation constraints. *Econometrica*, 57:615–635, 1989.
10. E. Even-Dar, A. Kesselman, and Y. Mansour. Convergence time to nash equilibria. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 502–513. Springer, 2003.
11. A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *PODC*, pages 347–351, 2003.
12. A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *STOC*, pages 604–612. ACM, 2004.

13. J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Hardness results for multicast cost sharing. *Journal of Public Economics*, 304(1-3):215–236, 2003.
14. J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proceedings of 32nd ACM Symposium on Theory of Computing (STOC)*, pages 218–227. ACM, 2000.
15. D. Fotakis, S. C. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. G. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2002.
16. M. X. Goemans, E. L. Li, V. S. Mirrokni, and M. Thottan. Market sharing games applied to content distribution in ad-hoc networks. In *MobiHoc*, pages 55–66. ACM, 2004.
17. M. X. Goemans, V. S. Mirrokni, and A. Vetta. Sink equilibria and convergence. In *FOCS*, pages 142–154. IEEE Computer Society, 2005.
18. K. Jain and V. V. Vazirani. Applications of approximation algorithms to cooperative games. In *STOC*, pages 364–372, 2001.
19. A. Kesselman, D. Kowalski, and M. Segal. Energy efficient communication in ad hoc networks from user's and designer's perspective. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(1):15–26, 2005.
20. E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 404–413. Springer, 1999.
21. M. Mavronicolas and P. G. Spirakis. The price of selfish routing. In *STOC*, pages 510–519, 2001.
22. I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13:111–124, 1996.
23. V. S. Mirrokni and A. Vetta. Convergence issues in competitive games. In *APPROX-RANDOM*, volume 3122 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 2004.
24. J. F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences*, volume 36, pages 48–49, 1950.
25. P. Penna and C. Ventre. More powerful and simpler cost-sharing methods. In *WAOA*, volume 3351 of *Lecture Notes in Computer Science*, pages 97–110. Springer, 2004.
26. P. Penna and C. Ventre. Sharing the cost of multicast transmissions in wireless networks. In *SIROCCO*, volume 3104 of *Lecture Notes in Computer Science*, pages 255–266. Springer, 2004.
27. P. Penna and C. Ventre. Free-riders in steiner tree cost-sharing games. In *SIROCCO*, volume 3499 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 2005.
28. R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.
29. T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of ACM*, 49(2):236–259, 2002.
30. L.S. Shapley. The value of n-person games. *Contributions to the theory of games*, pages 31–40, Princeton University Press, 1953.
31. A. Vetta. Nash equilibria in competitive societies, with applications to facility location, traffic routing and auctions. In *FOCS*, pages 416–425. IEEE Computer Society, 2002.

Very Sparse Leaf Languages

Lance Fortnow[1] and Mitsunori Ogihara[2,⋆]

[1] University of Chicago
fortnow@cs.uchicago.edu
[2] University of Rochester
ogihara@cs.rochester.edu

Abstract. Unger studied the balanced leaf languages defined via polylogarithmically sparse leaf pattern sets. Unger shows that NP-complete sets are not polynomial-time many-one reducible to such balanced leaf language unless the polynomial hierarchy collapses to Θ_2^p and that Σ_2^p-complete sets are not polynomial-time bounded-truth-table reducible (respectively, polynomial-time Turing reducible) to any such balanced leaf language unless the polynomial hierarchy collapses to Δ_2^p (respectively, Σ_4^p).

This paper studies the complexity of the class of such balanced leaf languages, which will be denoted by VSLL. In particular, the following tight upper and lower bounds of VSLL are shown:

1. coNP $\subseteq$ VSLL $\subseteq$ coNP/poly (the former inclusion is already shown by Unger).
2. coNP/1 $\not\subseteq$ VSLL unless PH $= \Theta_2^p$.
3. For all constant $c > 0$, VSLL $\not\subseteq$ coNP/n^c.
4. P/$(\log\log(n) + O(1)) \subseteq$ VSLL.
5. For all $h(n) = \log\log(n) + \omega(1)$, P/$h \not\subseteq$ VSLL.

1 Introduction

Bovet, Crescenzi, and Silvestri [2] introduced the concept of leaf languages — the languages defined in terms of the pattern appearing at the leaf-level a polynomial-time nondeterministic Turing machine that outputs a symbol along each computation path. The concept of using outputs of nondeterministic Turing machines for defining complexity classes appears earlier, in a paper by Goldschlager and Parberry [8], but it is in this work of Bovet, Crescenzi, and Silvestri that the concept was formulated and fully explored. Given a polynomial-time nondeterministic Turing machine M that accepts all inputs on all computations and that outputs a symbol from an alphabet Γ, leafstring$_M$ is the function that maps each input of M to the word produced by reading the output symbols in the computation tree of M on input x according to the dictionary order over the computation paths of M on x. Given a language K over the alphabet Γ, the leaf language with respect to M and K, is the set of all inputs x such that leafstring$_M(x) \in K$.

⋆ This work is support in part by XEROX/NYSTAR Grant C040130 and NSF-EIA Grant 0205061.

Bovet, Crescenzi, and Silvestri showed that many well-known complexity classes can be characterized this way using some simple leaf languages, including NP, coNP, and PP. The leaf languages offer a rich theory of complexity classes and are very strongly connected with branching programs [1] and bottleneck Turing machines [5].

Recently, Unger [15] studied the leaf languages defined with respect to poly-logarithmically sparse leaf pattern sets. We call this complexity class VSLL (Very Sparse Leaf Languages) as Unger did not give a name to this class. Unger showed that if SAT is polynomial-time many-one reducible to a language in VSLL then $\mathrm{PH} = \Theta_2^p$ and that if a Σ_2^p-complete set is polynomial-time bounded-truth-table reducible to a language in VSLL then $\mathrm{PH} = \Delta_2^p$.

The leaf languages defined with respect to such very sparse leaf pattern sets are related to polynomially sparse sets. Although he did not make it an explicit claim, Unger's proof of the former result essentially uses the fact that VSLL is included coNP/poly. Here coNP/poly is the "nonuniform-coNP" [11], in the sense that each language in the class is decidable in coNP with the aid of an "advice" string that is polynomially long and that is dependent solely on the input length. As we will show explicitly in this paper, for each language $L \in$ VSLL, the advice function that puts L in coNP/poly maps each input length to a polynomial number of members of L and the set of the words appearing in advice strings is a sparse subset of L.

We thus naturally question the connection between VSLL and sparse sets. Indeed, the proof of the former of the two results of Unger is reminiscent of the technique that uses the census function in NP exploited in the result of Kadin [10] that showed the existence of sparse Turing-complete sets for NP collapses the polynomial hierarchy to Θ_2^p; and the proof of the latter directly uses the left-set technique of Ogihara and Watanabe [13]. However, it is unclear whether the assumption $\mathrm{NP} \subseteq \mathrm{VSLL}$ for the first result implies the existence of sparse Turing-complete sets for NP. This observation motivates us to explore the complexity of VSLL.

It is easy to see that $\mathrm{coNP} \subseteq \mathrm{VSLL}$, via the poly-logarithmically sparse leaf pattern language $\{0^{2^n} \mid n \geq 0\}$ (see [15]). We show that this is in fact a tight lower bound of VSLL. The class coNP/1, i.e., coNP with a single-bit of advice, is not included in VSLL unless NP is already included in VSLL, which, according to the result of Unger, collapses the polynomial hierarchy to Θ_2^p. In fact, we show that for any recursive complexity class $\mathcal{C}$, if $\mathcal{C}/1 \subseteq \mathrm{VSLL}$ then $\mathcal{C} \subseteq \mathrm{P/poly}$. We also show a "provably" tight lower bound for VSLL with respect to polynomial-time decision with advice. While $\mathrm{P}/\log\log \subseteq \mathrm{VSLL}$, $\mathrm{P}/(\log\log(n) + \omega(1)) \not\subseteq$ VSLL.

Along with the above lower bound results, we show two tight upper bounds. First, if $\mathrm{VSLL} \subseteq \mathrm{P/poly}$ then PH collapses to Σ_2^p, more precisely, to class S_2^p [6,14]. Second, for an arbitrary constant $c > 0$, $\mathrm{VSLL} \not\subseteq \mathrm{coNP}/n^c$ holds with no assumption.

2 Preliminaries

Let Σ be the alphabet $\{0,1\}$. As usual, Σ^* denotes the set of all words over Σ and $\mathcal{N}$ denotes the set of all natural numbers. For each $n \in \mathcal{N}$, Σ^n denotes the set of all words over Σ having length n. For each $n \in \mathcal{N}$ and for each language L, $L^{=n}$ denotes $\{x \mid x \in L \wedge |x| = n\}$ and $L^{\leq n}$ denotes $\{x \mid x \in L \wedge |x| \leq n\}$. For a language A and for a natural number n, $\mathrm{census}_A(n) = \| L^{\leq n} \|$. The function census_A is called the *census function* of A.

We assume the reader's familiarity in basic complexity classes and reducibility notions, including classes P, NP, coNP, PH, $\{\Sigma_k^p, \Pi_k^p, \Delta_k^p, \Theta_k^p\}_{k \geq 0}$, L, and NL and reducibility notions $\leq_m^p$, $\leq_T^p$, and $\leq_{btt}^p$.

Let M be a polynomial time-bounded nondeterministic Turing machine transducer such that M accepts on all inputs and along all computation paths and such that on each computation path M outputs a symbol from an alphabet Γ. We call such a machine M an NP *character transducer*. For each input x and for each computation path π on M on x, let $M_\pi(x)$ be the output of M on x along path π. For each input x, define

$$\mathrm{leafstring}_M(x) = M_{\pi_1}(x) \cdots M_{\pi_r}(x),$$

where $\pi_1, \ldots, \pi_r$ is the enumeration in the dictionary order of all computation paths of M on input x. By abuse of notation, for a set of inputs S, we define

$$\mathrm{leafstring}_M(S) = \{\mathrm{leafstring}_M(x) \mid x \in S\}.$$

For $A \subseteq \Gamma^*$, let $\mathrm{Leaf}_M^{\mathrm{P}}(A)$ be the language consisting of all inputs x such that $\mathrm{leafstring}_M(x) \in A$.

Definition 1. *For a language $A \subseteq \Gamma^*$, the language class* $\mathrm{Leaf}^{\mathrm{P}}(A)$ *is the set of all languages* $\mathrm{Leaf}_M^{\mathrm{P}}(A)$*, where M is an* NP *character transducer.*

For a language class $\mathcal{C}$, the language class $\mathrm{Leaf}^{\mathrm{P}}(\mathcal{C})$ *is* $\{\mathrm{Leaf}^{\mathrm{P}}(A) \mid A \in \mathcal{C}\}$.

We pay a special attention to the leaf language defined via a Turing machine whose computation tree is *balanced* in the following sense: there exists a polynomial p such that for all inputs x, the computation tree of the Turing machine on input x is the full complete binary tree of height $p(|x|)$. We call such a machine a *balanced-tree* NP *character transducer*. We will use $\mathrm{BalancedLeaf}^{\mathrm{P}}(A)$ and $\mathrm{BalancedLeaf}^{\mathrm{P}}(\mathcal{C})$, respectively, to denote $\mathrm{Leaf}^{\mathrm{P}}(A)$ and $\mathrm{Leaf}^{\mathrm{P}}(\mathcal{C})$ in which the NP character transducers are restricted to be balanced-tree character transducers.

2.1 Sparse Sets

A language S is *sparse* if there exists a polynomial p such that for all n it holds that $\mathrm{census}_S(n) \leq p(n)$. By SPARSE we denote the set of all sparse sets.

A language S is *poly-logarithmically sparse* (or *polylog sparse*, for short) if there exist positive constants c and d such that for all n $\mathrm{census}_S(n) \leq c(\log n)^d$. By PLOGSPARSE we denote the set of all poly-logarithmically sparse sets.

2.2 Advice Classes

Let f be a function from $\mathcal{N}$ to itself and let $\mathcal{C}$ be a complexity class. Then $\mathcal{C}/f$ is the set of all languages L for which there exist $A \in \mathcal{C}$ and a function h such that

– for all x, $|h(x)| \le f(|x|)$, and
– for all x, $x \in L$ if and only if $\langle x, h(0^{|x|})\rangle \in A$.

Complexity classes defined this way are called "advice" classes and the function h is called an "advice" function. By poly we denote the set of all polynomials. It is a well-known fact that $\mathrm{P/poly} = \mathrm{P}^{\mathrm{SPARSE}}$ [11].

2.3 Very Sparse Leaf Languages

We define VSLL (Very Sparse Leaf Languages) to be the set of all leaf languages whose leaf pattern sets are poly-logarithmically sparse.

Definition 2. $\mathrm{VSLL} = \mathrm{BalancedLeaf}^{\mathrm{P}}(\mathrm{PLOGSPARSE})$.

Unger [15] showed the following:

Theorem 1 (Unger).

1. *If* $\mathrm{NP} \le_m^p \mathrm{VSLL}$ *then* $\mathrm{PH} = \Theta_2^p$.
2. *If* $\Sigma_2^p \le_{btt}^p \mathrm{VSLL}$ *then* $\mathrm{PH} = \Delta_2^p$.
3. *If* $\Sigma_2^p \le_T^p \mathrm{VSLL}$ *then* $\mathrm{PH} = \Sigma_4^p$.

3 Fundamental Properties of VSLL

We show below a number of properties of the class VSLL. The proofs in this section are omited due to page limit, but can be found in the full version of the paper [7].

First, the class VSLL is closed under $\le_m^p$-reductions; that is, if $A \le_m^p B$ and $B \in \mathrm{VSLL}$ then $A \in \mathrm{VSLL}$.

Proposition 1. VSLL *is closed under* $\le_m^p$*-reductions.*

Next, we present a property that is explicitly shown by Unger [15].

Proposition 2 (Unger). $L \in \mathrm{VSLL}$ *if and only if there exist a sparse set* S *and a balanced-tree* NP *character transducer* M *such that for all* x,

$$x \in L \iff (\exists y \in S^{=|x|})[\mathrm{leafstring}_M(x) = \mathrm{leafstring}_M(y)].$$

A language A is $\le_{ctt}^{\mathrm{NP}}$-reducible to a language B if there exists a polynomial time-bounded nondeterministic Turing machine transducer N with the property that, on each input x,

– along each computation path of N on x, N produces a nonempty collection of words, $(y_1, \ldots, y_k)$; and

– $x \in A$ if and only if there exists a computation path π of N on x such that every word in the collection produced along π is a member of B.

Theorem 2. *If* $L \in \mathrm{VSLL}$ *then* $L \in \mathrm{coNP/poly}$ *with an "advice" function that is computable in polynomial time using an oracle that is* $\leq_{ctt}^{\mathrm{NP}}$*-reducible to* L*.*

The corollary follows immediately from the above theorem.

Corollary 1. *If* $\mathrm{NP} \subseteq \mathrm{VSLL}$ *then* $\mathrm{NP} \subseteq \mathrm{coNP/poly}$ *with an "advice" function computable in* Δ_2^p*.*

4 Lower Bounds of VSLL

Next we explore lower bounds of VSLL.

Proposition 3. $\mathrm{SPARSE} \subseteq \mathrm{VSLL}$*.*

Proof. Let S be an arbitrary sparse set. Define M to be a nondeterministic Turing machine that on input x behaves as follows: M nondeterministically selects $\pi \in \Sigma^{|x|}$, and then outputs 1 if $\pi = x$ and 0 otherwise. For each x, let r_x denote the lexicographic order of x in $\Sigma^{|x|}$. Then, for every x, $\mathrm{leafstring}_M(x) = 0^{r_x - 1}10^{2^{|x|} - r_x}$. Define $K = \{0^{r_x - 1}10^{2^{|x|} - r_x} \mid x \in S\}$. Then $K \in \mathrm{PLOGSPARSE}$. Thus, (M, K) witnesses that $S \in \mathrm{VSLL}$. ■

Proposition 4. $\mathrm{coNP} \subseteq \mathrm{VSLL} \subseteq \mathrm{coNP/poly}$*.*

Proof. The inclusion $\mathrm{VSLL} \subseteq \mathrm{coNP/poly}$ follows from Theorem 2. The inclusion $\mathrm{coNP} \subseteq \mathrm{VSLL}$ is shownby by Unger [15]. ■

It immediately follows from the above proposition that: if A is Turing-reducible to a language in VSLL then $A \in \Delta_2^p/\mathrm{poly}$. In the case where A is Σ_2^p-complete, the set A being Turing-reducible to a language in VSLL implies that $\Sigma_2^p \subseteq \Delta_2^p/\mathrm{poly}$, Then, by invoking Yap's Theorem [16] as presented in [4], we have the collapse $\mathrm{PH} = (\mathrm{S}_2^p)^{\mathrm{NP}}$, where S_2^p is the symmetric-Σ_2^p class [6,14] and is known to reside between $\mathrm{NP} \cup \mathrm{coNP}$ and $\mathrm{ZPP}^{\mathrm{NP}}$ [3].

Proposition 5. *If* $\Sigma_2^p \subseteq \mathrm{P}^{\mathrm{VSLL}}$ *then* $\mathrm{PH} = (\mathrm{S}_2^p)^{\mathrm{NP}}$.

The above result improves part 3 of Theorem 1.

Theorem 3. *Suppose* $\mathrm{coNP/1} \subseteq \mathrm{VSLL}$*. Then* $\mathrm{NP} \subseteq \mathrm{VSLL}$ *and* $\mathrm{NP} \subseteq \mathrm{P/poly}$*.*

Proof. Suppose that $\mathrm{coNP/1} \subseteq \mathrm{VSLL}$. For each infinite bit sequence $\boldsymbol{a} = (a_0, a_1, a_2, \cdots)$, define a language $Q[\boldsymbol{a}]$ as follows: For each x,

$$x \in Q[\boldsymbol{a}] \iff (a_{|x|} = 1 \vee x \notin \mathrm{SAT}).$$

Then, for every $\boldsymbol{a}$, $Q[\boldsymbol{a}] \in \mathrm{coNP/1}$, and thus, $Q[\boldsymbol{a}] \in \mathrm{VSLL}$ by our supposition.

Take $\boldsymbol{a}$ to be the characteristic function of a bi-immune set. Then, no recursive function can compute infinitely many bits of $\boldsymbol{a}$. This bi-immune requirement is met by choosing $\boldsymbol{a}$ be Kolmogorov random infinite sequence [12]. Suppose $Q[\boldsymbol{a}] \in$ VSLL is witnessed by a balanced-tree NP character transducer M and $K \in$ PLOGSPARSE. Let p be a polynomial such that $p(n)$ is the height of the computation tree of M. Let $c, d > 0$ be constants such that K is $c(\log n)^d$ sparse. Define $q(n)$ to be the polynomial $c(\log(2^{p(n)}))^d = c(p(n))^d$.

For each n, let

$$W_1(n) = \{ \text{leafstring}_M(x) \mid |x| = n \wedge x \in \text{SAT} \} \text{ and}$$
$$W_0(n) = \{ \text{leafstring}_M(x) \mid |x| = n \wedge x \notin \text{SAT} \}.$$

Note that, for all n,

1. if $a_n = 0$ then $W_1(n) \cap W_0(n) = \emptyset$ and $\| W_0(n) \| \leq q(n)$, and
2. if $a_n = 1$ then $\| W_0(n) \cup W_1(n) \| \leq q(n)$.

Assume that there are infinitely many n for which

$$\text{either } W_1(n) \cap W_0(n) \neq \emptyset \text{ or } \| W_0(n) \cup W_1(n) \| > q(n).$$

Let D be a deterministic Turing machine that, on input n,

– deterministically simulates M on all inputs of length n to check whether one of the above two conditions holds, and then
– outputs 1 if the former condition holds (since, by (1) in the above, it must be the case that $a_n = 1$), 0 if the latter condition holds (since, by (2) in the above, it must be the case that $a_n = 0$), and "?" otherwise.

This machine D correctly computes a_n for infinitely many n. This contradicts the assumption that $\boldsymbol{a}$ is Kolmogorov random. So, for all but finitely many n,

– $W_1(n) \cap W_0(n) = \emptyset$ and
– $\| W_0(n) \cup W_1(n) \| \leq q(n)$.

By making changes for a finite number of input lengths, M can be made to satisfy these conditions for all n. Define $K' = \cup_{n \geq 0} W_1(n)$. Then, K' is polylog sparse and (M, K') witnesses that SAT $\in$ VSLL.

To prove that the same assumption implies that NP is in P/poly, note that, for each n and for each pair (u, v) of distinct words in $W_0(n) \cup W_1(n)$, there is a bit position j at which u and v disagree. Such a position corresponds to a computation path of M on inputs of length n. Select such a position for each distinct pair from $W_0(n) \cup W_1(n)$. For each $u \in W_0(n) \cup W_1(n)$, by examining the output of $M(u)$ for the selected paths, leafstring$_M(u)$ can be uniquely distinguished. The number of such pairs is bounded by $(q(n))^2/2$. Since each position corresponds to a computation path of M, the list of all these positions and how the words $W_0(n) \cup W_1(n)$ can be distinguished using the list can be described using a polynomially long string. This implies that SAT $\in$ P/poly. ∎

The following corollary follows from the above theorem by applying Theorem 1.

Corollary 2. *If* coNP/1 $\subseteq$ VSLL *then* PH $= \Theta_2^p$.

Note that, in the proof of the part that shows NP $\subseteq$ P/poly in the above theorem, the fact that the decision for Q given $\boldsymbol{a}$ is in coNP is never used. So, we can generalize the proof and show the following:

Theorem 4. *For every reasonable class $\mathcal{C}$ of recursive sets, $\mathcal{C}/1 \subseteq$ VSLL implies $\mathcal{C} \subseteq$ P/poly.*

Corollary 3. *If* (NP $\cap$ coNP)/1 $\subseteq$ VSLL *then* NP $\cap$ coNP $\subseteq$ P/poly.

Corollary 4. *If* UP/1 $\subseteq$ VSLL *then* UP $\subseteq$ P/poly.

In the following, define $\log\log(0) = \log\log(1) = 0$.

Proposition 6. P/$(\log\log + O(1)) \subseteq$ VSLL.

Proof. Let $L \in$ P/$(\log\log + O(1))$ via $A \in$ P and a function $f(n)$. We can assume that there exists an integer k such that, for all n, $|f(0^n)| \leq \lceil \log\log(n) + k \rceil$. Define $\mu(n) = \sum_{i=0}^{\lceil \log\log(n)+k \rceil} 2^i$. Then, for each n $\mu(n)$ is the number of possibilities for the "advice" string at length n and $\mu(n) \leq 2^{\log\log(n)+k+2}$. Let p be an arbitrary polynomial such that $2^{p(n)} \geq \mu(n)$. Let M be a nondeterministic Turing machine that on input x behaves as follows:

1. M nondeterministically guesses $y \in \Sigma^{p(|x|)}$ and computes its lexicographic order j of y in $\Sigma^{p(|x|)}$.
2. M outputs 0 if $j > \mu(n)$.
3. M computes the word w that has the lexicographic order j in Σ^*.
4. M outputs 1 if $\langle x, w \rangle \in A$ and 0 otherwise.

Clearly, M can be made to run in polynomial time. For each x, leafstring$_M(x)$ has length $2^{p(|x|)}$ and satisfies the following:

- for every j, $1 \leq j \leq \mu(|x|)$, the j-th bit of leafstring$_M(x)$ is 1 if (x is in L in the case where the j-th word is the "advice" string) and 0 otherwise, and
- for every j, $\mu(|x|) + 1 \leq j \leq 2^{p(|x|)}$, the j-th bit of leafstring$_M(x)$ is 0.

For each n, let j_n be the lexicographic order of the "advice" string for length n. Let K_n be the set of all words w having length $2^{p(n)}$ such that the j_n-th bit of w is 1 and such that for every j, $\mu(n) + 1 \leq j \leq 2^{p(n)}$, the j-th bit of w is 0. Define $K = \cup_{n \geq 0} K_n$.

We claim that the membership of L in VSLL is witnessed by M and K. For all x, $x \in L$ if and only if leafstring$_M(x) \in K_{|x|}$. To show that $K \in$ PLOGSPARSE, note that, for each n, $\mu(n) \leq 2^{\log\log(n)+k+2}$, $\| K_n \| \leq 2^{\mu(n)}$, and each word in K_n has length $2^{p(n)}$. Let $N = 2^{p(n)}$. Then, we have $n \leq p(n) = \log(N)$, so $\mu(n) \leq 2^{\log\log\log(N)+k+2} = 2^{k+2}(\log\log(N))$, and thus, $2^{\mu(n)} \leq 2^{2^{k+2}(\log\log(N))} = (\log N)^{2^{k+2}}$, which implies that $K \in$ PLOGSPARSE. This proves the proposition. ■

Theorem 5. *For all functions $h(n) = \log\log(n) + \omega(1)$, $\mathrm{P}/h \not\subseteq \mathrm{VSLL}$.*

Proof. Let $h(n)$ be a polynomial-time computable function such that $h(n) = \log\log(n) + \omega(1)$. Without loss of generality, we may assume that $h(n)$ is monotonically nondecreasing and $h(n) \leq \log(n)$ for all n.

For each infinite bit sequence $\boldsymbol{a} = (a_0, a_1, a_2, \cdots)$, define $Q[\boldsymbol{a}]$ as follows. Divide $\boldsymbol{a}$ into blocks of consecutive bits having length $h(0), h(1), h(2), \ldots$. More precisely, the first $h(0)$ bits of $\boldsymbol{a}$ (that is, $a_0, \ldots, a_{h(0)-1}$) become the first block, the next $h(1)$ bits (that is, $a_{h(1)}, \ldots, a_{h(0)+h(1)-1}$) become the second block, the next $h(2)$ bits (that is, $a_{h(1)+h(2)}, \ldots, a_{h(0)+h(1)+h(2)-1}$) become the third block, and so on. For each n, let B_n be the n-th block. For each n, we define $Q[\boldsymbol{a}]^{=n}$, the length-n portion of $Q[\boldsymbol{a}]$, as follows: Let $B_n = (d_1, \ldots, d_{h(n)})$ and let $D = 1 + \sum_{i=1}^{h(n)} d_i 2^{i-1}$. Then, $Q[\boldsymbol{a}]^{=n}$ is the set of all $x \in \Sigma^n$ whose D-th bit is a 1. For every $\boldsymbol{a}$, $Q[\boldsymbol{a}] \in \mathrm{P}/h$.

Assume, by way of contradiction, that $\mathrm{P}/h \subseteq \mathrm{VSLL}$. Take $\boldsymbol{a}$ to be a sequence with the following property:

(*) For every exponential-time deterministic Turing machine transducer M, if for all but finitely many n M on input 0^n outputs a word of length $h(n)$, then for infinitely many n, B_n is equal to the output of M on input 0^n.

A sequence $\boldsymbol{a}$ with this property can be constructed by simple diagonalization, but it suffices to take $\boldsymbol{a}$ to be a Kolmogorov random sequence relative to h.

Suppose that $Q[\boldsymbol{a}] \in \mathrm{VSLL}$ is witnessed by a balanced-tree NP character transducer M and a polylog sparse K. We then construct a deterministic algorithm that for all but finitely many n computes an $h(n)$-bit pattern ρ_n such that $B_n \neq \rho_n$.

By our assumption, $Q[\boldsymbol{a}] \in \mathrm{VSLL}$, so there exists a polynomial p such that for all n it holds that

$$\| \operatorname{leafstring}_M(Q[\boldsymbol{a}]^{=n}) \| \leq p(n).$$

Let n be sufficiently large. Let $B_n = (d_1, \ldots, d_{h(n)})$ and let $D = 1+\sum_{i=1}^{h(n)} d_i 2^{i-1}$. The range of the integer D is 1 to $\Delta = 2^{h(n)}$. For each i, $1 \leq i \leq \Delta$, and for each $b \in \{0, 1\}$, let

$$W_b(i) = \{\operatorname{leafstring}_M(x) \mid x \in \Sigma^n \wedge \text{ the } i\text{-th bit of } x \text{ is } b\}.$$

Suppose that there exists an i, $1 \leq i \leq \Delta$, such that $W_0(i) \cap W_1(i) \neq \emptyset$. Let i be such one. There exist $y, z \in \Sigma^n$ such that the i-th bit of y is 0, the i-th bit of z is 1, and $\operatorname{leafstring}_M(y) = \operatorname{leafstring}_M(z)$. This means that either both y and z are members of $Q[\boldsymbol{a}]^{=n}$ or both y and z are nonmembers of $Q[\boldsymbol{a}]^{=n}$. If D were i, then only z would be a member of $Q[\boldsymbol{a}]^{=n}$. So, it must be the case that $D \neq i$. Thus, if there exists an i, $1 \leq i \leq \Delta$, such that $W_0(i) \cap W_1(i) \neq \emptyset$, then it holds that $B_n \neq \beta_i$, where β_i is the word in $\Sigma^{h(n)}$ whose rank is i; that is, $\| \{w \in \Sigma^{h(n)} \mid w \leq \beta_i\} \| = i$. So, we take the smallest such i and set ρ_n to β_i.

We claim that there is always such an i. Assume, by way of contradiction, that for all i, $1 \leq i \leq \Delta$, $W_0(i) \cap W_1(i) = \emptyset$. Divide Σ^n according the first Δ

bits. For each $u \in \Sigma^{\Delta}$, let $w(u) = \mathrm{leafstring}_M(u0^{n-\Delta})$. By our assumption, for all u, u' in Σ^{Δ} $u \neq u' \Rightarrow w(u) \neq w(u')$. So, regardless of what the value of D is, for exactly half of $u \in \Sigma^{\Delta}$, $u0^{n-\Delta} \in Q[\boldsymbol{a}]^{=n}$. This implies that

$$\| \mathrm{leafstring}_M(Q[\boldsymbol{a}]^{=n}) \| \geq 2^{\Delta-1}.$$

Then, by our supposition about $h(n)$, we have

$$\begin{aligned} \| \mathrm{leafstring}_M(Q[\boldsymbol{a}]^{=n}) \| &\geq 2^{\Delta-1} \\ &= 2^{2^{h(n)}-1} \\ &= 2^{2^{\log\log(n)+\omega(1)}-1} \\ &= 2^{\log(n)^{\omega(1)}-1}. \end{aligned}$$

The last quantity is super-polynomial, that is, a function that grows faster than any polynomial, in particular, faster than $p(n)$. This contradicts our supposition that $\| \mathrm{leafstring}_M(Q[\boldsymbol{a}]^{=n})x) \| \leq p(n)$. Thus, the claim holds.

Define T to be a machine that, on input 0^n, executes the above algorithm and outputs ρ_n. It is not hard to see that M runs in time 2^{cn} for some constant c. For all but finitely many n, M on input 0^n outputs a word having length $h(n)$ not equal to B_n. This contradicts the property (*) of $\boldsymbol{a}$. Hence, $Q[\boldsymbol{a}] \notin$ VSLL. This proves the theorem. ■

5 Upper Bounds of VSLL

Next we prove upper bounds of VSLL.

Since coNP $\subseteq$ VSLL and P/poly is closed under complementation, by the Karp–Lipton Theorem [11] (see [3,4]), we have the following:

Proposition 7. VSLL $\subseteq$ P/poly *then* PH $= \mathrm{S}_2^p$.

Also, we show that the upper bound coNP/poly $\supseteq$ VSLL is a tight one.

Theorem 6. *For any constant c,* VSLL $\not\subseteq$ REC$/n^c$*, where* REC *is the class of all recursive sets.*

Proof. Let $c > 0$ be a fixed integer. Assume, by way of contradiction, that VSLL $\subseteq$ REC$/n^c$. By Proposition 3, SPARSE $\subseteq$ VSLL. We show that there is a sparse set not in REC$/n^c$.

Let $d = c + 2$. Let $\boldsymbol{a} = (a_0, a_1, a_2, \ldots)$ be a Kolmogorov random string. We construct a S from $\boldsymbol{a}$ as follows.

Fix an integer ν such that $2^{\nu}\nu^d \leq 2^{2\nu}$. As in the proof of Theorem 5, we divide $\boldsymbol{a}$ into blocks. This time the first block has length μ^{d+1}, the second block has length $(\mu+1)^{d+1}$, the third block has length $(\mu+2)^{d+1}$, and so. In other words, for each $i \geq 1$, the i-th block consist of the $(\mu+i-1)^{d+1}$ bits of a positioned between $\sum_{j=1}^{i-1}(\mu+j)^{d+1}$ and $(\sum_{j=1}^{i}(\mu+j)^{d+1}) - 1$. Then, for each $i \geq 1$, divide the i-th block, which has length $(\mu+i-1)^{d+1}$, into $(\mu+i-1)$ words of length $(\mu+i-1)^d$.

Let m be an arbitrary integer greater than or equal to ν. The $(m-\nu+1)$-st block of $\boldsymbol{a}$, which consists of m^d words of length m, defines $S \cap \Sigma^{2m}$ as follows. Let $w_1, \ldots, w_{m^d}$ be the m^d words. Let $b_1, \ldots, b_{m^d}$ be the rank of these words in Σ^m; that is, for each i, $1 \leq i \leq m^d$, $b_i = \| \{y \in \Sigma^m \mid y \leq w_i\} \|$. Then, for each i, $1 \leq i \leq m^d$, define z_i to be the word in Σ^{2m} whose rank is $b_1 + \cdots + b_i$. For all i, $1 \leq i \leq m^d$, $1 \leq b_i \leq 2^m$. Since $2^m m^d \leq 2^{2m}$, $z_1, \ldots, z_{m^d}$ are well defined and $z_1 < \cdots < z_{m^d}$. Also, the $(m-\nu+1)$-st block of $\boldsymbol{a}$ can be recovered from $z_1, \cdots, z_{m^d}$.

For all n, $\mathrm{census}_S(n) = \sum_{i:1\leq i\leq n}(i/2)^d \leq n^{d+1}/2^d$, so S is sparse, and thus, $S \in \mathrm{VSLL}$.

Since $\mathrm{VSLL} \subseteq \mathrm{REC}/n^c$ by our assumption, we have a halting Turing machine M and an advice function h such that

– for all n, $|h(0^n)| \leq n^c$, and
– for all x, $x \in S$ if and only if M on input $\langle x, h(0^{|x|})\rangle$ accepts.

For each even n, let $H_n = h(1)\#h(2)\#\cdots\#h(n)$. Let N be a machine that, on input w of the form $u_1\#u_2\#\cdots\#u_k$ such that $k \geq 2\nu$, k is even and $u_1, \ldots, u_k$ contain no $\#$, simulate M on all inputs of the form $\langle x, u_{|x|}\rangle$ such that $|x|$ is even and is greater than or equal to 2ν, and then recover the blocks of $\boldsymbol{a}$ corresponding to those x's accepted by M. Then, for all even $n \geq 2\nu$, N on input H_n produces the prefix of $\boldsymbol{a}$ having length $\sum_{i=\nu}^{n/2} i^d > n^d/2^d$. Since $d = c+2$, for all but finitely many n, the length of the prefix thus produced is greater than or equal to $n^{c+1.5}$.

On the other hand, the combination of machine N and the advice H_n can be encoded as an input word to a universal Turing machine having length $|H_n| + \alpha$, where α is a constant not depending on n. For all but finitely many n, $|H_n| \leq \sum_{i:1\leq i\leq n} n^c < n^{c+1}$.

This implies that for all but finitely many n, the prefix of $\boldsymbol{a}$ having length at least $n^{c+1.5}$ has Kolmogorov complexity at most n^{c+1}, which contradicts the assumption that $\boldsymbol{a}$ is Kolmogorov random. Thus, $S \notin \mathrm{REC}/n^c$. Hence, $\mathrm{VSLL} \not\subseteq \mathrm{REC}/n^c$. ■

6 Logarithmic Space-Bounded Character Transducers

We extend the concept of very sparse leaf languages by considering logarithmic space-bounded nondeterministic computation in place of polynomial time-bounded nondeterministic computation. Logarithmic space-bounded nondeterministic machines have polynomially many IDs. We often shrink their computation trees by identifying nodes corresponding to the same ID to one, but for defining leafstring, we will view the computation trees as binary trees without collapsing nodes. Let $\mathrm{Leaf}^{\log}$ and $\mathrm{BalancedLeaf}^{\log}$ respectively denote the $\mathrm{Leaf}^{\mathrm{P}}$ and $\mathrm{BalancedLeaf}^{\mathrm{P}}$ thus defined. Let $\mathrm{VSLL}^{\log}$ be the VSLL defined in terms of $\mathrm{BalancedLeaf}^{\log}$.

For a logarithmic space-bounded balanced-tree nondeterministic Turing machine character transducer M, whether two inputs x and y produce the same

leafstring_M can be tested as follows: Simulating M concurrently on the two inputs x and y choosing the same branch for both of them at each nondeterministic step. If such simulation arrives at an accepting state with two distinct output symbols for x and y, then $\text{leafstring}_M(x) \neq \text{leafstring}_M(y)$. If every simulation arrives at an accepting state with an identical output character for x and y, then $\text{leafstring}_M(x) \neq \text{leafstring}_M(y)$. So, the equality can be tested in coNL. Since NL is closed under complement, this implies that the test can be done in NL.

Theorem 7.

1. $\text{NL} \subseteq \text{VSLL}^{\log} \subseteq \text{NL/poly}$.
2. *If* $\text{NL}/1 \subseteq \text{VSLL}^{\log}$ *then* $\text{NL} \subseteq \text{L/poly}$.
3. *For all constants* $c > 0$, $\text{VSLL}^{\log} \not\subseteq \text{REC}/n^c$.
4. $\text{VSLL}^{\log} \supseteq \text{L}/(\log\log(n) + O(1))$.
5. $\text{VSLL}^{\log} \not\supseteq \text{L}/(\log\log(n) + \omega(1))$.

7 Conclusion

We would like to find a natural problem in VSLL. Graph Isomorphism is an obvious candidate because Graph Isomorphism is in coAM $\subseteq$ coNP/poly [9] but not known to lie in coNP. However a similar proof to Theorem 1 will show that NP$\cap$VSLL is low for Θ_2^p. Since Graph Isomorphism is in NP, Graph Isomorphism in VSLL implies Graph Isomorphism is low for Θ_2^p, which is not known to hold.

Acknowledgments

The authors would like to thank Lane Hemaspaandra and Falk Unger for useful discussions.

References

1. D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, **38**:150–164, 1989.
2. D. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science,* **104**(2):263–283, 1992.
3. J. Cai. $\text{S}_2^p \subseteq \text{ZPP}^{\text{NP}}$. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science,* pages 620–629. IEEE Computer Society Press, Los Alamitos, CA, 2001.
4. J. Cai, V. Chakaravarthy, L. Hemaspaandra, and M. Ogihara. Competing provers yield improved Karp–Lipton collapse results. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, pages 535–546, Springer-Verlag Lecture Notes in Computer Science 2607, 2003.
5. J. Cai and M. Furst. PSPACE survives constant-width bottlenecks. *International Journal on Foundations of Computer Science*, **2**(1):67–76, 1991.

6. R. Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters* **57**(5):237–241, 1996.
7. L. Fortnow and M. Ogihara. Very sparse leaf languages. Technical Report 899. Department of Computer Science, University of Rochester, Rochester, NY, June, 2006.
8. L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, **43**:43–58, 1986.
9. O. Goldreich, S. Micali and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, **38**(3):690–728, 1991.
10. J. Kadin. $\mathrm{P}^{\mathrm{NP}[\log n]}$ and sparse Turing-complete sets for NP. *SIAM Journal on Computing.* **17**(6):1263–1282, 1988. Erratum, **20**(2):404, 1991.
11. R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th Symposium on Theory of Computing*, pages 302–309, ACM Press, New York, NY, 1980.
12. M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its application.* Springer-Verlag, New York, NY, 1993.
13. M. Ogiwara and O. Watanabe. On polynomial time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal of Computing.* **20**(3):471–483, 1991.
14. A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity* **7**(2):152–162, 1998.
15. F. Unger. On small hard leaf languages. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, pages 781-792, Lecture Notes in Computer Science 3618, Springer-Verlag, 2005.
16. C.-K. Yap. Consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science* 26:287–300, 1983.

On the Correlation Between Parity and Modular Polynomials

Anna Gál[⋆] and Vladimir Trifonov

Dept. of Computer Science, University of Texas at Austin,
Austin, TX 78712-1188, USA
{panni, vladot}@cs.utexas.edu

Abstract. We consider the problem of bounding the correlation between parity and modular polynomials over $\mathbb{Z}_q$, for arbitrary odd integer $q \geq 3$. We prove exponentially small upper bounds for classes of polynomials with certain linear algebraic properties. As a corollary, we obtain exponential lower bounds on the size necessary to compute parity by depth-3 circuits of certain form. Our technique is based on a new representation of the correlation using exponential sums.

Our results include Goldmann's result [Go] on the correlation between parity and degree one polynomials as a special case. Our general expression for representing correlation can be used to derive the bounds of Cai, Green, and Thierauf [CGT] for symmetric polynomials, using ideas of the [CGT] proof. The classes of polynomials for which we obtain exponentially small upper bounds include polynomials of large degree and with a large number of terms, that previous techniques did not apply to.

1 Introduction

In this paper, we study the correlation between the MOD_2 function and Boolean functions computed by depth-2 circuits with a MOD_q gate at the top (for odd q), and AND gates at the input level (called $MOD_q \circ AND$ circuits). The Boolean function $MOD_m : \{0,1\}^n \to \{0,1\}$ is defined to be 0 when the sum of the input bits is divisible by m, and 1 otherwise. For every $MOD_m \circ AND$ circuit there is a multilinear polynomial P over $\mathbb{Z}_m$ such that on inputs $\mathbf{x} \in \{0,1\}^n$, the output of the circuit is 0 if and only if $P(\mathbf{x})$ is 0 modulo m. There is a straightforward way to associate such a polynomial with each circuit, using the inputs associated with the AND gates to form the monomials. This polynomial is called the *defining polynomial* of the circuit, and its degree is the largest fan-in of the AND gates in the circuit. Thus, depth-2 circuits of the above form correspond to polynomials over the ring $\mathbb{Z}_m$.

The *correlation* $C(f_1, f_2)$ between two Boolean functions $f_1, f_2 : \{0,1\}^n \to \{0,1\}$ is defined as $C(f_1, f_2) = \frac{1}{2^n} \sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f_1(\mathbf{x})} (-1)^{f_2(\mathbf{x})}$. Our interest in this question is motivated by its relevance to circuit complexity lower bounds. In addition, we believe that the question is also interesting on its own right.

⋆ Supported in part by NSF Grant CCF-0430695 and an Alfred P. Sloan Research Fellowship.

1.1 Bounded Depth Circuits

Proving lower bounds on the size of Boolean circuits for specific functions is one of the central problems in complexity theory. It is also considered to be notoriously difficult, since for example, superpolynomial lower bounds on the size of Boolean circuits computing a function from the complexity class NP would imply that $P \neq NP$. However, even much weaker (e.g. superlinear) lower bounds seem to remain out of reach of the current techniques. Imposing various restrictions on the circuits and developing lower bound methods for restricted circuit models has received a lot of attention in the last few decades. The hope is to extend such techniques, and develop new methods that are applicable towards stronger and stronger models.

One of the circuit models that has been extensively studied is bounded depth circuits. The results of [Aj, FSS, Ha, Yao85] show that the parity function cannot be computed by AC^0 circuits (constant depth polynomial size circuits with AND, OR, NOT gates). Barrington [Ba] defined the class $\mathrm{ACC}^0 = \cup_q \mathrm{ACC}^0(q)$, where $\mathrm{ACC}^0(q)$ denotes the class of constant depth, polynomial size circuits with AND,OR, NOT and MOD_q gates. Smolensky [Sm] proved that $MOD_r \notin \mathrm{ACC}^0(p^k)$ when p and r are distinct primes. The power of $\mathrm{ACC}^0(q)$ circuits when q is not a prime power is much less understood. For example, it is not known if all of NP can be computed by depth-3 $\mathrm{ACC}^0(6)$ circuits.

Depth-3 circuits can be surprisingly powerful. Allender [Al] proved that AC^0 is contained in the class of depth-3 circuits of quasipolynomial ($2^{(\log n)^{O(1)}}$) size with a $MAJORITY$ gate at the top, MOD_2 gates in the middle, and AND gates of $(\log n)^{O(1)}$ fan-in at the input level. (Such circuits are referred to as $MAJ \circ MOD_2 \circ AND_{(\log n)^{O(1)}}$ circuits.) Yao [Yao90] proved that ACC^0 is contained in the class of depth-3 threshold circuits of quasipolynomial ($2^{(\log n)^{O(1)}}$) size with AND gates of $(\log n)^{O(1)}$ fan-in at the input level. But it remains open if ACC^0 is contained in the class of quasipolynomial ($2^{(\log n)^{O(1)}}$) size $MAJ \circ MOD_q \circ AND_{(\log n)^{O(1)}}$ circuits, for some fixed q. In other words, it is not known whether Allender's result [Al] can be extended to ACC^0. (Essentially this question was asked by Green in [Gr02].) A recent result of Bourgain [Bo05] implies that parity requires exponential size $MAJ \circ MOD_q \circ AND_{\epsilon \log n}$ circuits, for any odd q and ϵ depending on q. It is not clear how to extend Bourgain's result to larger fan-in AND gates. The results of Håstad and Goldmann [HG] imply that a function in ACC^0 (the generalized inner product function) requires exponential size $MAJ \circ MOD_2 \circ AND_{O(\log n)}$ circuits. The results of Razborov and Wigderson [RW] imply that some function in ACC^0 requires $n^{\Omega(\log n)}$ size $MAJ \circ MOD_2 \circ AND$ circuits. This result was recently extended to circuits with arbitrary AC^0 circuits in place of the AND gates by Hansen and Miltersen [HM]. [RW] and [HM] build on the results of [HG]. However, the method in [HG] applies for arbitrary symmetric gates in the middle layer. Thus, in view of Yao's result [Yao90], these results cannot be directly extended to obtain exponential lower bounds for computing an ACC^0-function by $MAJ \circ MOD_q \circ AND_{(\log n)^{O(1)}}$ circuits.

Other combinations of threshold, MOD and AND gates in depth-3 circuits and other definitions of MOD gates have been also considered, and in some of these models exponential lower bounds have been proved for functions in ACC^0 (see e.g. [BM, Gro, GT, KP]). The power of $MAJ \circ MOD \circ AND_{(\log n)^{O(1)}}$ circuits remains less understood.

1.2 Correlation and Circuit Lower Bounds

Obtaining exponential lower bounds for $MAJ \circ MOD_q \circ AND$ circuits under various restrictions has received considerable attention in the last few years (e.g. [AB, CGT, Gr99, Gr02, Go]. The starting point of all these papers, including [HG] which considers the more general $MAJ \circ SYM \circ AND$ circuits, is the following special case of a lemma of [HMPST].

Lemma 1. *(Lemma 3.3, [HMPST]) Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function computed by a Boolean circuit with a fan-in m (unweighted) threshold gate on top, taking the results of the subcircuits $C_1, \ldots, C_m$ as inputs to the threshold gate. Let $g_i : \{0,1\}^n \to \{0,1\}$ be the Boolean function computed by the subcircuit C_i ($i = 1, \ldots, m$). Let f be a balanced function, i.e. $|f^{-1}(0)| = |f^{-1}(1)|$. Then for at least one of the subcircuits (for some $1 \le i \le m$), the absolute value of the correlation $|C(f, g_i)|$ is at least $1/m$.*

Thus, upper bounds on the absolute value of the correlation of a balanced function f with arbitrary functions that can be computed by circuits of a given class $\mathcal{C}$, imply lower bounds on the fan-in of the $MAJORITY$ gate in $MAJ \circ \mathcal{C}$ type circuits for computing f.

In particular, proving that the absolute value of the correlation of parity with modular polynomials over $\mathbb{Z}_q$ of certain type is exponentially small, implies exponential lower bounds on the size of the corresponding $MAJ \circ MOD_q \circ AND$ circuits. Smolensky's results [Sm] imply that for p and r distinct primes, the absolute value of the correlation of the MOD_r function and low degree polynomials over $\mathbb{Z}_{p^k}$ is at most $\frac{1}{n^{1/2-o(1)}}$. Note that the technique of [Sm] does not yield smaller bounds on the absolute value of the correlation even for degree 2 and very sparse polynomials, and it cannot be applied over $\mathbb{Z}_q$, if q is not a prime power. It is also curious to note that on the other hand, by Ajtai's [Aj] result we know that the absolute value of the correlation of parity with functions in AC^0 is exponentially small, and it remains exponentially small even allowing superpolynomial number of gates [Ha]. Cai, Green and Thierauf [CGT] proved that the absolute value of the correlation of parity with symmetric polynomials of degree $(\log n)^{O(1)}$ over $\mathbb{Z}_q$ for q odd, is exponentially small (at most $2^{-n^{\Omega(1)}}$). This was generalized by Green [Gr99] to proving similar exponentially small upper bounds on the absolute value of the correlation of the MOD_p function with symmetric polynomials of degree $(\log n)^{O(1)}$ over $\mathbb{Z}_q$ when p is a prime that does not divide q.

Extending these bounds to allowing non-symmetric polynomials posed a significant challenge. The degree 1 case was solved by Goldmann [Go], who proved that the absolute value of the correlation of MOD_p and MOD_q when p has a prime factor that does not divide q, is at most $2^{-\Omega(n)}$. Alon and Beigel [AB] showed that the absolute value of the correlation of parity with degree 2 polynomials over $\mathbb{Z}_q$ for odd q, is at most $2^{-(\log n)^{\epsilon}}$ for some constant $\epsilon < 1$, and for degree $O(1)$ polynomials the absolute value of the correlation is $o(1)$. Note that the bounds of [AB] are weaker than the $\frac{1}{n^{1/2-o(1)}}$ upper bounds implied by Smolensky's results [Sm], but [Sm] is applicable only when q is a prime power. The first improvement over the bounds of [Sm] and [AB] for non-symmetric polynomials of degree greater than 1 was achieved by Green [Gr02]. Green [Gr02] proved that the absolute value of the correlation of parity with degree 2 polynomials over $\mathbb{Z}_3$ is at most $2^{-\Omega(n)}$. The method used in [Gr02] very specifically relies on the degree being at most 2 and $q = 3$, and appears to be not applicable to other degrees or other values of q. A breakthrough was achieved by Bourgain [Bo05], proving that for q odd, and p, q relatively prime, the absolute value of the correlation between MOD_p and degree d polynomials over $\mathbb{Z}_q$ is exponentially small for $d < \epsilon \log n$, where ϵ depends on p and q. Bourgain's result was generalized by Green, Roy and Straubing [GRS] to arbitrary (not necessarily odd) q and p, q relatively prime.

While Bourgain's result resolves the question about the correlation between parity and modular polynomials of degree up to $\epsilon \log n$, it leaves open the question described in the previous section about whether Allender's result [Al] can be extended to ACC^0. To obtain sufficiently strong lower bounds for depth 3 circuits of the desired type by bounding correlation, we would need to be able to provide estimates on the correlation for up to polylogarithmic degree polynomials.

1.3 Our Approach

We suggest a new approach to estimate the correlation of parity with modular polynomials over $\mathbb{Z}_q$ that is applicable to arbitrary odd q, and provides improvements over the previous bounds for several classes of polynomials.

The starting point of our approach is a representation of the correlation using exponential sums. Exponential sums have been used to estimate correlation in several previous papers starting with the results of Cai, Green and Thierauf [CGT] for symmetric polynomials and also in [Gr99, Gr02, Bo05, GRS]. We give a representation of the correlation of parity with polynomials over $\mathbb{Z}_q$ using exponential sums in a very general setting. The novelty of our representation is that it allows to use certain linear algebraic properties of the terms of the corresponding polynomials. We also present a general expression for representing correlation, that can be used to yield our results as well as to derive the bounds of Cai, Green, and Thierauf [CGT] for symmetric polynomials, using ideas of the [CGT] proof. The two approaches can be viewed in a unifying framework as working with different components of our expression.

We are able to evaluate the exponential sums involved in this representation under various conditions, and we obtain exponentially small upper bounds on the absolute value of the correlation between parity and modular polynomials of certain type. Interestingly, the classes of polynomials for which we prove exponentially small bounds include polynomials of very large degree and polynomials with very large number of terms as well (as long as they satisfy some other, linear algebraic conditions). All previous methods assumed small degree to obtain exponentially small upper bounds on correlation with parity, thus could not be used to obtain our results. Moreover, some of our results yield exponentially small upper bounds on the absolute value of the correlation with parity over every nonempty subset of the variables.

Due to space limitations, all proofs are omitted from this extended abstract.

2 Exact Representations of the Correlation

2.1 Notation

For $r \in \mathbb{Z}^+$ and $z \in \mathbb{Z}$ define $\delta_r(z)$ to be 1, if $r|z$, and -1, otherwise. We will use $x \equiv_r y$ to denote $r|(x-y)$. $\equiv_r$ is extended to vectors by applying the congruence on every coordinate. That is, we use the notation $\mathbf{x} \equiv_r \mathbf{y}$ to indicate that $x_i \equiv_r y_i$ for every coordinate. The exponent function is extended to vectors in a component-wise manner, that is, $c^{\mathbf{x}}$ denotes $(c^{x_1}, \ldots, c^{x_n})$.

We will denote with $\mathbf{0}$ the all 0's vector, where the dimension of the vector will be understood by the context. Similarly $\mathbf{1}$ is the all 1's vector. We use $\mathbf{1}_n$ to denote the all 1's vector of length n, we omit indicating the length when it is clear from the context. Vectors will be assumed to be in a column form and $\mathbf{x}^T$ is the row vector corresponding to a column vector $\mathbf{x}$. Similarly M^T is the transpose of a matrix M. For two vectors $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{x}^T\mathbf{y}$ is the usual inner product of the two vectors, that is, $\mathbf{x}^T\mathbf{y} = \sum_i x_i y_i$. For a matrix M and a vector $\mathbf{x}$, $M\mathbf{x}$ is the product of M and $\mathbf{x}$. Unless indicated otherwise, all sums and products are over the integers $\mathbb{Z}$.

The following notation for sets will be used: $[r] = \{1, \ldots, r\}$, $[0, r] = \{0, \ldots, r\}$, and $[0, 1) = \{a \in \mathbb{R} : 0 \leq a < 1\}$.

Let $h : \mathbb{Z}^n \to \mathbb{Z}$ be an arbitrary integer valued function and let $\mathbf{g} \in \{0,1\}^n$. We use the following notation.

$$C(\mathbf{g}, h) := 2^{-n} \sum_{\mathbf{x} \in \{0,1\}^n} \delta_2(\mathbf{g}^T\mathbf{x})\delta_q(h(\mathbf{x}))$$

We wish to estimate how well $MOD_q \circ AND$ circuits approximate parity. Let $f : \{0,1\}^n \to \{0,1\}$ be the function computed by a $MOD_q \circ AND$ circuit, and let P_f be the defining polynomial of the circuit. Then $(-1)^{f(\mathbf{x})} = \delta_q(P_f(\mathbf{x}))$ for $\mathbf{x} \in \{0,1\}^n$, and with our notation, the correlation between parity and f is equal to $C(\mathbf{1}, P_f)$. In general, our methods apply to estimating the correlation for the parity over arbitrary subsets of the input bits. Thus, we are interested in estimating $C(\mathbf{g}, P)$ for a multilinear polynomial $P(x_1, \ldots, x_n)$ with integer coefficients and a vector $\mathbf{g} \in \{0,1\}^n$.

Notice that we do not identify $\{0,1\}$ with $\mathbb{Z}_2$, i.e. arithmetic with numbers from $\{0,1\}$ is done in $\mathbb{Z}$, unless indicated otherwise. For $M \in \{0,1\}^{m \times n}$, $\mathrm{rk}_2(M)$ denotes the rank of M over $\mathbb{Z}_2$.

2.2 Exponential Sums

Following [CGT, Gr99, Gr02], we use exponential sums to represent the correlation. We give a representation of the correlation of parity with modular polynomials by exponential sums for arbitrary degree and arbitrary odd $q \geq 3$. Moreover, our representation applies to parity taken over arbitrary subsets of the input variables.

Let $\omega_q = e^{2\pi \mathrm{i}/q} = \cos 2\pi/q + \mathrm{i} \sin 2\pi/q$, the principal q-th root of unity, and $\bar{\omega} = \omega^{-1}$, the complex conjugate of ω. We omit q from the subscript of ω when it is clear from the context. We have the following lemma, which gives an alternative expression for the correlation $C(\mathbf{g}, h)$ between integer valued functions and parity.

Lemma 2. *Let $h : \mathbb{Z}^n \to \mathbb{Z}$, $\mathbf{g} \in \{0,1\}^n$, and let $q \geq 3$ be an odd integer. Then*

$$C(\mathbf{g}, h) = -\nu + \frac{2^{-(n-1)}}{q} \sum_{t=0}^{q-1} \sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\mathbf{g}^T \mathbf{x}} \omega^{th(\mathbf{x})},$$

where ν is 0 if $\mathbf{g} \neq \mathbf{0}$, and 1 otherwise.

Definition 1. *For $t \in [0, q-1]$, $h : \mathbb{Z}^n \to \mathbb{Z}$, and $\mathbf{g} \in \{0,1\}^n$ define*

$$C_t(\mathbf{g}, h) = 2^{-n} \sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\mathbf{g}^T \mathbf{x}} \omega^{th(\mathbf{x})}.$$

Notice that, if $\mathbf{g} \neq \mathbf{0}$, by the triangle inequality applied to the expression in Lemma 2, there exists $t \in [0, q-1]$ such that $|C(\mathbf{g}, h)| \leq 2|C_t(\mathbf{g}, h)|$. Hence, if $\mathbf{g} \neq \mathbf{0}$ and we can obtain an exponentially small bound on $|C_t(\mathbf{g}, h)|$ for every $t \in [0, q-1]$, then we will have an exponentially small bound on $|C(\mathbf{g}, h)|$. We can show that the converse is also true in some sense: if $|C(\mathbf{g}, h+c)|$ is exponentially small for every $c \in [0, q-1]$, then $|C_t(\mathbf{g}, h)|$ is exponentially small as well for every $t \in [0, q-1]$.

2.3 Matrix Notation

Let $P(\mathbf{x})$ be a multilinear polynomial with integer coefficients. First we will construct a multilinear polynomial $Q(\mathbf{y})$ with integer coefficients and with the same degree as $P(\mathbf{x})$ such that, for $\mathbf{x} \in \{0,1\}^n$, $P(\mathbf{x}) \equiv_q Q((-1)^{\mathbf{x}})$. Recall that $(-1)^{\mathbf{x}}$ denotes $((-1)^{x_1}, \ldots, (-1)^{x_n})$.

For $q \geq 3$ odd, there exists a unique integer $\rho \in [q-1]$ such that $2\rho \equiv_q 1$. For $z \in \mathbb{Z}$, let $l(z) = \rho(1-z)$ and extend l to vectors in a component-wise manner, that is, $l(\mathbf{y}) = (\rho(1-y_1), \ldots, \rho(1-y_n))$. Notice that for $\mathbf{x} \in \{0,1\}^n$

$$\mathbf{x} \equiv_q l((-1)^{\mathbf{x}}). \tag{1}$$

Define $Q(\mathbf{y}) = P(l(\mathbf{y}))$. Since l (considered as a univariate polynomial) is linear with integer coefficients, Q is a multilinear polynomial with integer coefficients of the same degree as P. Also, by (1), for every $\mathbf{x} \in \{0,1\}^n$ we have $P(\mathbf{x}) \equiv_q Q((-1)^{\mathbf{x}})$. Thus

$$C(\mathbf{g}, P) = 2^{-n} \sum_{\mathbf{x} \in \{0,1\}^n} \delta_2(\mathbf{g}^T \mathbf{x}) \delta_q(Q((-1)^{\mathbf{x}})). \tag{2}$$

Our next goal will be to express $Q((-1)^{\mathbf{x}})$ using a linear transformation. Since Q is multilinear with integer coefficients, we can write it as $Q(\mathbf{y}) = \sum_{I \subseteq [n]} c_I y_I$, where $c_I \in \mathbb{Z}$ and $y_I = \prod_{i \in I} y_i$. Let $M \in \{0,1\}^{m \times n}$ be the matrix whose rows are the incidence vectors of the subsets $I \subseteq [n]$, each repeated ($c_I \bmod q$) times. (The incidence vector of the empty set is the all zero row, and $y_\emptyset = 1$ for any $\mathbf{y}$.) Notice that the degree of Q (and therefore the degree of P) is at most d if and only if M has at most d 1's per row. For $\mathbf{x} \in \{0,1\}^n$ we have

$$Q((-1)^{\mathbf{x}}) \equiv_q \mathbf{1}^T (-1)^{M\mathbf{x}}. \tag{3}$$

We use the following notation:

$$C(\mathbf{g}, M) := 2^{-n} \sum_{\mathbf{x} \in \{0,1\}^n} \delta_2(\mathbf{g}^T \mathbf{x}) \delta_q(\mathbf{1}^T (-1)^{M\mathbf{x}}).$$

Then, using (2) and (3), we obtain the following.

Lemma 3. *Let P and Q be multilinear polynomials with integer coefficients such that $P(\mathbf{x}) \equiv_q Q((-1)^{\mathbf{x}})$ for $\mathbf{x} \in \{0,1\}^n$. Let M correspond to Q according to the above mapping, and let $\mathbf{g} \in \{0,1\}^n$. Then $C(\mathbf{g}, P) = C(\mathbf{g}, M)$*

Before we proceed, consider the following example. Let $Q(\mathbf{y}) = \prod_{i=1}^n y_i - 1$. Then M consists of a single row of all 1's, and $q-1$ copies of the all zero row. The corresponding correlation is $C(\mathbf{1}, M) = 1$, since $\delta_q(\mathbf{1}^T (-1)^{M\mathbf{x}}) = \delta_q(\prod_{i=1}^n (-1)^{x_i} + q - 1) = \delta_2(\mathbf{1}^T \mathbf{x})$ for every $\mathbf{x} \in \{0,1\}^n$ (and every $q \geq 3$).

Definition 2. *For $t \in [0, q-1]$, $M \in \{0,1\}^{m \times n}$, and $\mathbf{g} \in \{0,1\}^n$ define*

$$C_t(\mathbf{g}, M) = 2^{-n} \sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\mathbf{g}^T \mathbf{x}} \omega^{t \mathbf{1}^T (-1)^{M\mathbf{x}}}.$$

With this notation, using Lemma 2

$$C(\mathbf{g}, M) = -\nu + \frac{2}{q} \sum_{t=0}^{q-1} C_t(\mathbf{g}, M), \tag{4}$$

where ν is 0 if $\mathbf{g} \neq \mathbf{0}$, and 1 otherwise.

Remark 1. Our methods directly apply to estimating the correlation between parity and $MOD_q \circ MOD_2$ circuits. In this case, given a $MOD_q \circ MOD_2$ circuit, there is a multilinear polynomial Q with integer coefficients such that on inputs $\mathbf{x} \in \{0,1\}^n$ the output of the circuit is 0 if and only if $Q((-1)^{\mathbf{x}})$ is 0 modulo q. Let M be the matrix corresponding to the polynomial Q as above. Then the correlation between the output of the circuit and the parity of the subset of variables corresponding to the vector $\mathbf{g}$ is equal to $C(\mathbf{g}, M)$.

2.4 Main Lemma

It is immediate from (4) by the triangle inequality that for $\mathbf{g} \neq \mathbf{0}$, $|C(\mathbf{g}, M)|$ is exponentially small if $|C_t(\mathbf{g}, M)|$ is exponentially small for every $t \in [0, q-1]$.

Thus, we will be concerned with giving bounds on $|C_t(\mathbf{g}, M)|$, and use them to bound $|C(\mathbf{g}, M)|$ using (4).

Definition 3. *For $M \in \{0,1\}^{m\times n}$ and $\mathbf{g} \in \{0,1\}^n$, define*

$$I(M) = \left\{\mathbf{z} \in \{0,1\}^n : \exists\, \mathbf{y} \in \{0,1\}^m \text{ s.t. } M^T\mathbf{y} \equiv_2 \mathbf{z}\right\},$$
$$K(M, \mathbf{g}) = \left\{\mathbf{y} \in \{0,1\}^m : M^T\mathbf{y} \equiv_2 \mathbf{g}\right\}.$$

The following lemma is our main technical tool for obtaining bounds on the correlation based on linear algebraic properties of the polynomials.

Lemma 4. *Let $t \in [0, q-1]$, $M \in \{0,1\}^{m\times n}$, and $\mathbf{g} \in \{0,1\}^n$. Then*

$$C_t(\mathbf{g}, M) = 2^{-m} \sum_{\mathbf{y}\in K(M,\mathbf{g})} (\omega^t - \bar{\omega}^t)^{|\mathbf{y}|}(\omega^t + \bar{\omega}^t)^{m-|\mathbf{y}|},$$

where $|\mathbf{y}|$ is the number of 1's in $\mathbf{y}$.

2.5 A More General Framework

Definition 4. *For $\mathbf{g} \in \{0,1\}^n$, $A \in \mathbb{N}^{m\times n}$, and $\mathbf{b} \in \mathbb{N}^m$, define*

$$\kappa(\mathbf{g}, A, \mathbf{b}) = \sum_{\mathbf{x}\in\{0,1\}^n : A\mathbf{x}=\mathbf{b}} (-1)^{\mathbf{g}^T\mathbf{x}}.$$

For $\mathbf{z} = (z_1, \ldots, z_m)$ define the following polynomial over $z_1, \ldots, z_m$.

$$T(\mathbf{g}, A, \mathbf{z}) = \sum_{\mathbf{b}\in I_A} \kappa(\mathbf{g}, A, \mathbf{b}) z_1^{b_1} \cdot \ldots \cdot z_m^{b_m},$$

where $I_A = \{\mathbf{b} \in \mathbb{N}^m : 0 \le b_i \le \sum_{j=1}^n a_{ij},\ for\ i \in [m]\}$.

First note that this definition includes as a special case the definition of Krawtchouk polynomials [Sz]. To see this take $m = 1$ and A to be an all 1's row of length n. Then $\kappa(\mathbf{g}, \mathbf{1}_n^T, k) = K_k^{(n)}(|\mathbf{g}|)$, where $K_k^{(n)}(l)$ is the k-th Krawtchouk polynomial, i.e. $K_k^{(n)}(l) = \sum_{i=0}^k (-1)^i \binom{l}{i}\binom{n-l}{k-i}$ which is the coefficient of y^k of the polynomial $(1-y)^l(1+y)^{n-l}$.

In our definition $\kappa(\mathbf{g}, A, \mathbf{b})$ is not necessarily a polynomial except in special cases, but it gives the coefficient of the monomial $z_1^{b_1} \cdot \ldots \cdot z_m^{b_m}$ of the polynomial $T(\mathbf{g}, A, \mathbf{z})$, which can be written in the following form.

$$T(\mathbf{g}, A, \mathbf{z}) = \prod_{j\in[n]:g_j=1} \left(1 - \prod_{i\in[m]} z_i^{a_{ij}}\right) \prod_{j\in[n]:g_j=0} \left(1 + \prod_{i\in[m]} z_i^{a_{ij}}\right). \tag{5}$$

(This expression can be verified by expanding the right side of (5) and then grouping the terms in $z_1, \ldots, z_m$ of the same form together.) Thus, in some sense the functions $\kappa(\mathbf{g}, A, \mathbf{b})$ are analogues of the Krawtchouk polynomials in a more general setting.

As we have seen before, for $\mathbf{g} \neq \mathbf{0}$, and arbitrary $h : \mathbb{Z}^n \to \mathbb{Z}$, the correlation $|C(\mathbf{g}, h)|$ is exponentially small if $|C_t(\mathbf{g}, h)|$ is exponentially small for every $t \in [0, q-1]$. We give the following general expression for $C_t(\mathbf{g}, h)$.

Lemma 5. *Let $q, r \in \mathbb{N}^+$ and $t \in [0, q-1]$. Let $h : \mathbb{Z}^n \to \mathbb{Z}$ be such that there exists $A \in \mathbb{N}^{m \times n}$ and $G : \mathbb{N}^m \to \mathbb{Z}$ such that for every $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{z} \in \mathbb{N}^m$, if $\mathbf{z} \equiv_r A\mathbf{x}$, then $h(\mathbf{x}) \equiv_q G(\mathbf{z})$. Then*

$$C_t(\mathbf{g}, h) = 2^{-n} \sum_{\mathbf{y} \in [0,r-1]^m} T(\mathbf{g}, A, \omega_r^{\mathbf{y}}) \phi_{r,q,t}(\mathbf{y}, G) , \tag{6}$$

where $\phi_{r,q,t}(\mathbf{y}, G) = r^{-m} \sum_{\mathbf{z} \in [0,r-1]^m} \bar{\omega}_r^{\mathbf{y}^T \mathbf{z}} \omega_q^{tG(\mathbf{z})}$.

This lemma can be used to derive our main lemma (Lemma 4) that we use to exploit the linear algebraic properties of the polynomials when estimating correlation, as well as the bounds of Cai, Green and Thierauf [CGT] for symmetric polynomials. Interestingly, the statement yields these two arguments by working with different parts of the expression. To obtain our results in this paper we carefully estimate $\phi_{r,q,t}$, but we set things up so that for T we only have one possible nonzero value, and we just have to argue about when is T nonzero. To obtain the bounds of [CGT], we carefully estimate T, and use only a trivial bound on $\phi_{r,q,t}$, namely that $|\phi_{r,q,t}| \leq 1$. The key to derive the bounds of [CGT] from Lemma 5 is to show that for symmetric polynomials the matrix $A = \mathbf{1}_n^T$ with only one row and certain small odd r have the desired properties.

Note that all our expressions so far have been precise and we obtained exact representations of the correlation between parity and modular polynomials. Next we consider cases where we can obtain exponentially small upper bounds on the absolute value of our expressions.

3 Bounds Based on the Linear Algebraic Structure of the Polynomials

Lemma 4 allows us to obtain estimates on the correlation of the polynomial $P(\mathbf{x})$ with parity, based on the linear algebraic properties of the matrix $M \in \{0,1\}^{m \times n}$ considered as a matrix over $\mathbb{Z}_2$. Recall that to obtain M, first P is transformed to another polynomial Q, and the rows of M are defined by the terms of Q as described in Section 2.3. Also note that our methods can be used to estimate the correlation of modular polynomials and parity over arbitrary subsets of the variables. Parity is taken over the coordinates that are 1 in the vector $\mathbf{g}$, taking parity of all the variables corresponds to using $\mathbf{g} = \mathbf{1}$.

An immediate consequence of Lemma 4 is that $C_t(\mathbf{g}, M) = 0$, if $K(M, \mathbf{g}) = \emptyset$. Hence if $\mathbf{0} \neq \mathbf{g} \notin I(M)$, then $C(\mathbf{g}, M) = 0$. Thus we get the following interesting statement.

Theorem 1. *Let P and Q be multilinear polynomials with integer coefficients such that $P(\mathbf{x}) \equiv_q Q((-1)^{\mathbf{x}})$ for $\mathbf{x} \in \{0,1\}^n$. Let M be the matrix corresponding to Q, and let $\mathbf{g} \in \{0,1\}^n$. If the rows of the matrix M do not span the vector $\mathbf{g}$ over $\mathbb{Z}_2$, that is when $\mathbf{g} \notin I(M)$, then the correlation $C(\mathbf{g}, P) = 0$.*

This theorem extends the well known fact that if a polynomial P does not depend on all the variables over which we take parity, then the correlation between parity and P is zero.

Estimating $C_t(\mathbf{g}, M)$ when $\mathbf{g} \in I(M)$ is a challenging task in general. We prove that $|C_t(\mathbf{g}, M)|$ is exponentially small for certain classes of matrices M. Then, (4) can be used to obtain upper bounds on $|C(\mathbf{g}, M)|$. Note that while our estimates of $|C_t(\mathbf{g}, M)|$ apply to arbitrary $\mathbf{g} \in \{0,1\}^n$, and give good bounds even for $\mathbf{g} = \mathbf{0}$, the bounds on $|C(\mathbf{g}, M)|$ are interesting only for $\mathbf{g} \neq \mathbf{0}$, since in (4) $\nu = 1$ for $\mathbf{g} = \mathbf{0}$. Notice that if $\mathbf{g} \neq \mathbf{0}$, then $C_0(\mathbf{g}, M) = 0$, so it is enough to estimate $|C_t(\mathbf{g}, M)|$ for $t \in [q-1]$.

First we consider the class of non-singular matrices over $\mathbb{Z}_2$.

Theorem 2. *Let $M \in \{0,1\}^{n \times n}$ be a non-singular matrix over $\mathbb{Z}_2$, and $\mathbf{g} \in \{0,1\}^n$. Let $q \geq 3$ be an odd integer, and let $t \in [q-1]$. There exists $\gamma = \gamma(q) \in [0,1)$ (depending only on q) such that $|C_t(\mathbf{g}, M)| \leq \gamma^n$.*

It is interesting to note that Theorem 2 gives exponentially small upper bounds on the absolute value of the correlation with parity for polynomials possibly with arbitrarily large degree that previous techniques did not apply to. It is also interesting that we get exponentially small correlation with respect to parity over arbitrary nonempty subsets of the variables. On the other hand, Theorem 2 does not apply for example to all degree one polynomials, since repeating rows (according to the coefficients of Q) means that the matrix is singular. We are able to extend our results to a much larger class of matrices, that also includes all degree one polynomials. First we consider an extension of the non-singularity condition, next we state our results with respect to arbitrary matrices that have a partition into submatrices with not too much overlap between the subspaces spanned by their rows.

Definition 5. *A matrix $M \in \{0,1\}^{m \times n}$ is* block non-singular *over $\mathbb{Z}_2$ if M can be partitioned into submatrices $M_1, \ldots, M_k$ with $M_i \in \{0,1\}^{m_i \times n}$ for $i \in [k]$, such that $\sum_i^k rk_2(M_i) = rk_2(M) = n$.*

Note that the above definition implies that the linear subspaces over $\mathbb{Z}_2$ spanned by the rows of the different blocks are disjoint, except containing the $\mathbf{0}$ vector. In other words, the row-space of the matrix M is the direct sum of the row-spaces of the submatrices in the partition.

Theorem 3. *Let $M \in \{0,1\}^{m \times n}$ be a block non-singular matrix over $\mathbb{Z}_2$. Let $q \geq 3$ be an odd integer, and let $t \in [q-1]$. Given $\mathbf{g} \in \{0,1\}^n$, let $\ell(\mathbf{g})$ be the smallest number of blocks in the partition that contribute a nonzero vector to obtaining $\mathbf{g}$ as a linear combination over $\mathbb{Z}_2$ of the rows of M. There exists $\gamma = \gamma(q) \in [0,1)$ (depending only on q) such that $|C_t(\mathbf{g}, M)| \leq \gamma^{\ell(\mathbf{g})}$.*

Note that if M corresponds to an arbitrary degree one polynomial, then M is block non-singular, and $\ell(\mathbf{g}) = |\mathbf{g}|$ for any $\mathbf{g} \in \{0,1\}^n$. Thus, the above theorem contains Goldmann's result [Go] on the correlation between parity and degree one polynomials as a special case.

Given a matrix M, the bounds on the correlation obtained by Theorem 3 depend via $\ell(\mathbf{g})$ on over which subset of variables the parity is taken. On the other hand, Theorem 3 can be extended to yield exponentially small bounds on the correlation as long as the subspaces over $\mathbb{Z}_2$ spanned by the blocks do not overlap too much and many blocks are needed to span $\mathbf{g}$, that is, when $\sum_{i=1}^{k} rk_2(M_i) - rk_2(M)$ is relatively small and $\ell(\mathbf{g})$ is relatively large.

If we further restrict the class of polynomials and allow only coefficients relatively prime to q or 0 modulo q, we obtain a statement that gives the same (potentially exponentially small) upper bound on the absolute value of the correlation with the parity of *every* nonempty subset of the variables.

We say that a matrix $M \in \{0,1\}^{m \times n}$ is *nontrivial* if the polynomial Q it represents is not identically 0 modulo q over $\{-1,1\}^n$. Typically we are only interested in estimating $C_t(\mathbf{g}, M)$ for nontrivial M. Moreover, we can assume without loss of generality that every submatrix formed by a subset of the rows of M is nontrivial: deleting the rows of a trivial submatrix cannot change the value of $C_t(\mathbf{g}, M)$.

Theorem 4. *Let $M \in \{0,1\}^{m \times n}$, and $\mathbf{g} \in \{0,1\}^n$. Assume that every submatrix formed by a subset of the rows of M is nontrivial. Let $M_1, \ldots, M_k$ be an arbitrary partition of the nonzero rows of M into blocks, and let $r = max_{i \in [k]} rk_2(M_i)$. Let $q \geq 3$ be an odd integer, and let $t \in [q-1]$. Assume that M corresponds to a polynomial such that all coefficients are either relatively prime to q or 0 modulo q. Then there exists $\gamma = \gamma(q, r) \in [1/2, 1)$ (depending only on q and r) such that $|C_t(\mathbf{g}, M)| \leq 2^{\sum_{i=1}^{k} rk_2(M_i) - rk_2(M)} \gamma^k$.*

If q is prime, the extra condition we consider does not impose any restrictions, but for composite q it is essential: we have an example of a $\pmod{15}$ polynomial that otherwise satisfies the conditions of the theorem, but has coefficients not relatively prime to 15 and has constant correlation with parity.

Having $rk_2(M_i) \leq r$ for each block is not essential, it just makes the theorem simpler to state. It is enough for getting exponentially small upper bounds that a large number of blocks has small rank. Note that this holds for example if M is block non-singular with sufficiently many blocks, and in this case the correlation with parity over *every* nonempty subset of variables is exponentially small.

References

[Aj] M. Ajtai. Σ_1^1-formulae on finite structures. *Ann. Pure Appl. Logic* 24, 1983, 1-48.

[Al] E. Allender. A note on the power of threshold circuits. *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989, 580–584.

[AB] N. Alon, R. Beigel. Lower bounds for approximations by low degree polynomials over $\mathbb{Z}_m$, Proceedings of the 16th Annual IEEE Conference on Computational Complexity, 2001, 184–187

[Ba] D. Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC_1. *Journal of Computer and System Sciences* 38 (1989), 150-164.

[BM] R. Beigel, A. Maciel. Upper and lower bounds for some depth-3 circuit classes, in *Proceedings of the 30th Symposium on Foundations of Computer Science*, 1989, 580–584.

[Bo05] J. Bourgain. Estimation of certain exponential sums arising in complexity theory, *C.R. Acad. Sci. Paris* Ser. I 340 (2005) pp. 627-631.

[CGT] J.-Y. Cai, F. Green, and T. Thierauf. On the correlation of symmetric functions. *Mathematical Systems Theory* 29 (1996), 245–258.

[FSS] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial hierarchy. *Mathematical Systems Theory* 17, 1984, 13–27.

[Gr99] F. Green. Exponential sums and circuits with single threshold gate and mod-gates , *Theory Computing Systems* 32 (1999), 453-466.

[Gr02] F. Green. The Correlation between parity and quadratic polynomials mod 3, *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, 2002, 65–72.

[GRS] F. Green, A. Roy, H. Straubing. Bounds on an exponential sum arising in boolean circuit complexity, *C.R. Acad. Sci. Paris* Ser. I 340 (2005).

[Go] M. Goldmann. A note on the power of majority gates and modular gates, *Information Processing Letters* **53** (1995), 321–327

[Gro] V. Grolmusz. A weight-size tradeoff for circuits with mod m gates, in *Proceedings of the 26th ACM Symposium on the Theory of Computing*, 1994, 68–74.

[GT] V. Grolmusz, G. Tardos. Lower bounds for $MOD_p - MOD_m$ circuits, *SIAM J. Comput.*, 29, No. 4, 2000, 1209–1222.

[HMPST] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán. Threshold Circuits of bounded depth, *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, 1987, 99-110

[HM] K. Hansen and P. Miltersen. Some meet-in-the-middle circuit lower bounds. in Proceedings of MFCS, 2004, 334–345.

[Ha] J. Håstad. *Computational Limitations of Small-Depth Circuits.* MIT Press, 1986.

[HG] J. Håstad and M. Goldmann. On the power of small depth threshold circuits. *Computational Complexity* 1(2), 1991, 113–129.

[KP] M. Krause, P. Pudlák. On the computational power of depth 2 circuits with threshold and modulo gates, in *Proceedings of the 26th ACM Symposium on the Theory of Computing*, 1994, 48–57.

[LN] R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications, Cambridge University Press.

[RW] A. Razborov and A. Wigderson. $n^{\Omega(\log n)}$ Lower bounds on the size of depth-3 threshold circuits with AND gates at the bottom. *Information Processing Letters* 45, 1993, 303–307.

[Sm] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity, *Proceedings of the 19th Annual ACM Symposium on Theory of Computi ng*, 1987, 77–82

[Sz] G. Szegö. Orthogonal Polynomials. American Mathematical Society, 1939.

[Yao85] A. Yao. Separating the polynomial hierarchy by oracles. *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, 1985, 1–10.

[Yao90] A. Yao. On *ACC* and threshold circuits. *Proceedings of the 31th Annual IEEE Symposium on Foundations of Computer Science*, 1990, 619–627.

[Zb] S. Zabek. *Sur la périodicité modulo m des suites de nombres* $\binom{n}{k}$. Ann. Univ. Mariae Curie Sklodowska, A10 (1956), pp. 37–47.

Optimally Fast Data Gathering in Sensor Networks

Luisa Gargano and Adele A. Rescigno

Dip. di Informatica ed Applicazioni, Universitá di Salerno, 84081 Baronissi, Italy

Abstract. Efficient data gathering in sensor network is an important challenge. In this paper we address the problem of gathering sensed data to the sink of a sensor network minimizing the time to complete the process. We present optimal time data gathering algorithms for any type of topology of a sensor network. Our results improve on existing approximation algorithms. We approach the gathering problem by obtaining optimal solutions to the collision-free paths coloring problem.

1 Introduction

In this paper we study the following path coloring problem.

Collision–free paths from s in G: Given a graph $G = (V, E)$, a source node s, and a set of colors $\{1, 2, \cdots, |E||V|\}$, we want to establish in G for each node u a path connecting s to u and to color the edges of such paths so that: The colors of the edges of each fixed path are strictly increasing when moving along the path away from s; the coloring is collision–free, that is, no two edges sharing a common node are assigned the same color, even on paths leading to two different nodes. The goal is to minimize the value of the largest used color.

An example of an instance of the above problem is given in Fig. 1.

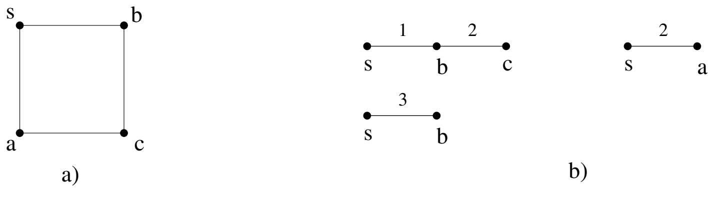


Fig. 1. a) A graph G; b) Collision-free paths from s in G using 3 colors

Our interest in the above problem arose from a gathering problem in sensor networks. The gathering problem and its relation to collision free path coloring are described in the next section.

Gathering in sensor networks. Sensor networks constitute an important class of emerging networked systems for applications in industries, transportation, manufacturing, environmental oversight, safety and security. They are multi-hop wireless networks formed by a large number of low-cost sensor nodes with limited battery power and processing capacity. Because of the nonmobile nature of the sensor nodes, as well as their finite energy resources (and therefore lifetime), there is a bootstrap phase in which the nodes self-organize to form the network. Information processing is allowed in a sensor network by merging sensing, signal processing, and communication functions.

One of the most important communication primitives that has to be provided by a sensor network is *data gathering*, i.e., information collected at sensors has to be sent to a *sink* node, which is responsible for further processing for end-user queries. Data gathering in sensor networks received much attention in the last few years, c.f.r. the surveys [1,3]. Most of the researches in the area focused on the problem of gathering reducing the energy consumption [5,7,12]. However, an other important factor to consider in data gathering applications is the *delay* of the gathering process. Indeed, the data collected by a node can frequently change thus making essential that they are received by the sink as soon as it is possible without being delayed by collisions [14].

Here we are concerned with efficiency limits of data gathering with respect to the time. Performances will be valued considering the network model already adopted in [6]. The sensor network is formally defined as a finite collection of identical nodes; a special node s will represent the sink.

Following [6], we assume that each sensor is equipped with an half–duplex interface, hence it cannot receive and transmit at the same time. Moreover, each node is equipped with *directional antennas* allowing the transmission over a distance R. This implies that for any given node in the network, we can individuate its neighbors as those sensors within distance R from it, that is, within its transmission/interference range. The use of directional antennas allows to select the neighbor to which the transmission is actually directed [6].

In this model, a *collision* happens at a node x if two or more of its neighbors try to transmit to x at the same time. However, simultaneous transmissions among pair of nodes may occur whenever the interference model is respected. The time is slotted so that one–hop transmission of one data item consumes one time slot, the network is assumed to be synchronous. Moreover, it is assumed that the only traffic in the network is due to sensor data, thus data transmissions can be completely scheduled.

Summarizing, the network can be represented by means of a graph where nodes represent the sensors and an edge exists between two nodes if the two sensors are in the range of each other; the collision–free data gathering problem can be then stated as follows [14].

> **Data Gathering.** Given a graph $G = (V, E)$ and a sink s, for each $v \in V - \{s\}$, schedule the multi-hop transmission of the data item sensed at v to s so that the whole process is collision–free and the time when the last data is received by s is minimized.

It follows that, under the above model, solving the collision–free path coloring problem is equivalent to solve the above scheduling problem for gathering in sensor networks. Indeed, let C denote the largest color used by an algorithm for the collision–free path coloring problem, a gathering schedule with delay C consists in scheduling a transmission from sensor y to x during slot t, with $1 \leq t \leq C$, iff the algorithm generates a path $(s, \dots, x, y, \dots)$ in which the edge (x, y) is assigned color $C - t + 1$.

Our results. In this paper we give optimal solutions for the collision–free path problem for any connected graph G. Namely, we will prove the following main result.

Theorem 1. *For any connected graph $G = (V, E)$ and $s \in V$, there exists an efficient algorithm to obtain an optimal solution to the collision-free paths problem from s in G.*

By the considerations made in the above section and Theorem 1 we have

Corollary 1. *Let $G = (V, E)$ be a graph representing any sensor network and let $s \in V$ be the sink. There exists an efficient algorithm to obtain an optimal solution to the data gathering problem to s in G.*

Related work. Data gathering is a fundamental issue in the context of sensor networks and several efficient data gathering algorithms have been proposed [1,3]. Most of them deal with the problem of maximize the lifetime of the sensor network through topology aware placement [5], efficient data flow [7], or data aggregation (when allowed by the typology of sensed data) [8].

The problem of minimizing the time, needed to complete data gathering, has also been recently recognized and studied. The work which is most related to the problem we consider in this paper is [6]. The authors of [6] use the same model for the sensor network adopted in this paper. The main difference with our work is that they deal with the case in which a node can have any number of data packet to be delivered to the sink. Under this assumption, [6] gives optimal collision–free strategies for trees and presents an approximation algorithm with performance ratio 2 for general networks. In the model of [6] we obtain optimal algorithms for any type of networks in the gathering hypothesis that each node has one packet of sensed data to deliver.

Another closely related work is [14]. It studies collision-free data gathering, delivering one data packet from each node to the sink. The difference with the model in [6] is the assumption of the possibility to have multiple channels between adjacent nodes. By adopting this model an approximation algorithm with performance ratio 2 is obtained.

In [2] gathering of data packets to the sink is considered under the condition that when a node transmits one data packet, all the nodes within a fixed distance d_T can receive while nodes within distance d_I ($d_I \geq d_T$) cannot listen to other transmissions due to interference (in our paper $d_I = d_T$ and a transmitting node can choose the neighbor to which transmit). Under the described model, lower bounds on the time to gather are given and NP-hardness is proved. An approximation algorithm with approximation factor 4 is also presented.

Papers [10,13] consider the time needed to gather in conjunction with the energy spent to complete the process. They present schemes that attempt to optimize the energy $\times$delay cost function.

Finally, we notice that several papers deal with broadcasting in wireless networks [11], however this problem is not the reverse of data gathering whereby different data packets are gathered to the sink.

Paper Overview. The rest of the paper is organized as follows: In Section 2 we formally describe the collision-free path coloring problem. Results for trees are reported in Section 3. Sections 4 and 5 present the proposed optimal solutions in case of biconnected graphs and general connected graphs, respectively.

2 Collision–Free Path Coloring: Problem Statement

Let $G = (V, E)$ be a weighted connected graph. The node $s \in V$ is a special node that will be called the *source*. Each node $u \in V - \{s\}$ is associated with an integer weight r_u that will be called the *request* of u. The set $\mathbf{r} = \{r_u \mid u \in V - \{s\}\}$ will represent the set of requests of the nodes in V.

Let $\mathcal{P}$ be any set of paths in G from s to (not necessarily all distinct) nodes in V. A *coloring* of $\mathcal{P}$ is an assignment of colors to the edges of the paths in $\mathcal{P}$. Colors will be chosen in the set of integers $\{1, 2, \ldots |E||V|\}$. Namely, a coloring of a path $\mathbf{p} \in \mathcal{P}$ is an assignment of a color $L(e, \mathbf{p})$ to each edge e of $\mathbf{p}$. Notice that any edge $e \in E(G)$ can belong to zero, one or more paths in $\mathcal{P}$.

Definition 1. *A* collision–free (CF) coloring L *of a set of paths* $\mathcal{P}$ *satisfies*

1) *for any path* $\mathbf{p} = (e_1, e_2, \ldots, e_h) \in \mathcal{P}$*, the coloring of the edges in* $\mathbf{p}$ *is strictly increasing, that is* $L(e_1, \mathbf{p}) < L(e_2, \mathbf{p}) < \cdots < L(e_h, \mathbf{p})$*,*
2) *for any* $v \in V$*, edges in* $E_v = \{(u, v) \in E \mid (u, v)$ *is an edge of a path in* $\mathcal{P}\}$*, get pairwise different colors, that is, for any* $\mathbf{p}, \mathbf{p}' \in \mathcal{P}$ *and* $(u, v), (u', v) \in E_v$ *(possibly,* $u = u'$ *or* $\mathbf{p} = \mathbf{p}'$*) it holds* $L((u, v), \mathbf{p}) \neq L((u', v), \mathbf{p}')$*.*

Definition 2. *An* instance of CF coloring *is a triple* $(G, \mathbf{r}, s)$ *where* G *is the graph,* s *is the source, and* $\mathbf{r}$ *is the set of requests.*

A solution *for* $(G, \mathbf{r}, s)$ *is a pair* $(\mathcal{P}, L)$ *where:* $\mathcal{P}$ *is a set of* r_u *paths (not necessarily distinct) from* s *to* u *in* G*, for any* $u \in V - \{s\}$*;* L *is a CF–coloring of* $\mathcal{P}$*. An* optimal solution *is a solution that minimizes the largest color (i.e. integer) assigned to any edge.*

We denote by $\mathcal{T}^*(G, \mathbf{r}, s)$ the value attained by the optimal solution $(\mathcal{P}^*, L^*)$ for $(G, \mathbf{r}, s)$[1].

Example. Let G be a ring with nodes $\{0, 1, 2, 3\}$, let $s = 0$, and $r_1 = 4, r_3 = 1, r_2 = 0$. An optimal solution for $(G, \mathbf{r}, 0)$ is the pair $(\mathcal{P}^*, L^*)$ where:
$\mathcal{P}^* = \{\mathbf{p}_3 = (0, 3), \mathbf{p}_1^0 = (0, 3, 2, 1), \mathbf{p}_1^i = (0, 1)$, for $i = 1, 2, 3\}$ and L^* is such that $L^*((0, 3), \mathbf{p}_3) = 4$, $L^*((0, 3), \mathbf{p}_1^0) = 2$, $L^*((3, 2), \mathbf{p}_1^0) = 3$, $L^*((2, 1), \mathbf{p}_1^0) = 4$, $L^*((0, 1), \mathbf{p}_1^i) = 2i - 1$, for $i = 1, 2, 3$.

[1] Notice that minimizing the largest color is equivalent to minimize the number of used colors. Indeed, we can limit ourselves to consider solutions where all colors in $\{1, \cdots, \mathcal{T}^*\}$ are used.

The following lower bounds on the optimal number of colors are given in [6].

Theorem 2. *[6] Given any instance $(G, \mathbf{r}, s)$,*

i) $\mathcal{T}^*(G, \mathbf{r}, s) \geq \sum_{u \in V-\{s\}} r_u$*;*

ii) $\mathcal{T}^*(G, \mathbf{r}, s) \geq \max_{v \in V-\{s\}} d(s, v) + r_v - 1 + 2 \sum_{u \in V_v} r_u$*,* *where $d(s, v)$ is the length of a shortest path between s and v in G and V_v is the set of all nodes different from v that result disconnected from s if v is removed from G together with all its incident edges.*

3 Trees

In this section we present an algorithm to solve the coloring problem for an instance $(T, \mathbf{r}, s)$ on a tree T, where the requests of the nodes are arbitrary positive integers, that is, $r_u \geq 1$ for each $u \in V(T) - \{s\}$. Such an algorithm was proposed in [6] (including the case $r_u = 0$) and obtains an optimal solution. We restate it here in the case of positive requests since it will be used to derive our optimal coloring algorithm for general graphs.

3.1 Notation

Definition 3. *Let T be any tree. We shall denote by $|T|$ the* size of T *in terms of* the number of requests *of the nodes in T, that is $|T| = \sum_{u \in V(T)} r_u$.*

Obviously, if $r_u = 1$ for each $u \in V(T)$ then $|T|$ is the number of nodes in T.

Definition 4. *Let T be a tree rooted at node s, and let $T_1, T_2, \cdots, T_m$, $m \geq 2$, be the subtrees of T rooted at the children of s in T. W.l.o.g. we assume that $|T_1| \geq |T_i|$ for $i \neq 1$. The tree T is called* balanced *, if $|T_1| \leq |T_2| + \cdots + |T_m| + 1$.*

In the following we shall use the following terminology.

– *s serves a node $u \in V - \{s\}$*: if a path from s to u is established together with its coloring;
– A node $u \in V - \{s\}$ *has been satisfied*: if r_u paths from s to u have been established together with their colorings;
– *s serves subtree T_i*: s serves a node u in T_i which is the furthest among all nodes in T_i which are not jet satisfied.
– a subtree T_i *has been satisfied*: if each node in T_i has been satisfied.

Definition 5. *Let $\mathbf{p}_u = (e_1, \cdots, e_h)$ be the path in a tree T from s to a node u. For any integer $k \geq 1$, the k*-increasing coloring of $\mathbf{p}_u$ *is the coloring $\hat{L}$ of the edges of $\mathbf{p}_u$ such that $\hat{L}(e_j, \mathbf{p}_u) = k + j - 1$ for $j = 1, \ldots, h$.*

An increasing coloring of $\mathbf{p}_u$ *is a k-increasing coloring of $\mathbf{p}_u$ for some $k \geq 1$.*

3.2 The CF-Coloring Algorithm on Trees

Unless otherwise stated we assume in the following that $T_1, T_2, \cdots, T_m$ are the subtrees of T rooted at the children of s.

The CF coloring algorithm is given in Fig. 2. The feasibility and the optimality of the solution given by the algorithm TREE-coloring were proved in [6] for general sets of requests. However a closed form of the optimal value was not given in [6]. We derive some (novel) properties of increasing colorings of trees and the exact value of the solution in the cases of our interest. Such results will be used to prove the optimality of the algorithm for general graphs.

Lemma 1. *Let u be any node with $u \neq s$. The largest color assigned to the edge (f,u), from the father of u in T to u is larger than any of the colors assigned to the edges in the subtree of T rooted at u.*

Corollary 2. *The largest color required by the algorithm TREE-coloring is always attained by one of the colors assigned to an edge from the root s to one of its neighbors.*

Theorem 3. *Let $r_u \geq 1$ for each $u \in V - \{s\}$. The optimal solution returned by the algorithm TREE-coloring on T has maximum color value equal to*

$$\mathcal{T}^*(T, \mathbf{r}, s) = \begin{cases} \sum_{i=1}^{m} |T_i| & \text{if } T \text{ is balanced} \\ 2|T_1| - 1 & \text{otherwise} \end{cases} \tag{1}$$

```
TREE-coloring $(T, \mathbf{r}, s)$
• Set $\mathcal{P} = \emptyset$, $k = 1$, and $g = 0$   [s served $T_g$ at the previous step, 0 means none]
  Set $R = \{1, 2, \cdots, m\}$   [R is the set of indices of subtrees not yet satisfied]
• while $R \neq \emptyset$
      Execute the following Step $k$
       if there exists $T_i$, with $j \neq g$, that is not satisfied then
           - Let $T_i$, $i \neq g$, be the one with the largest number of remaining requests.
           - $s$ serves $T_i$; let $u$ be the node which is served.
           - Color, using the $k$-increasing coloring $\hat{L}$, the path $\mathbf{p}_u$ from $s$ to $u$ in $T$,
           - Put $\mathcal{P} = \mathcal{P} \cup \{\mathbf{p}_u\}$.
           - If $T_i$ is satisfied then put $R = R - \{i\}$ .
           - Set $g = i$
       else [s remains idle] Set $g = 0$.
       Put $k = k + 1$.
• return $(\mathcal{P}, \hat{L})$
```

Fig. 2. The tree coloring algorithm

4 Biconnected Graphs

Consider a biconnected graph $G = (V, E)$. We shall first prove that it is always possible to find a spanning tree T rooted at the source s which is balanced (see

Definition 4), whenever $\mathbf{r} = \{r_u = 1 \mid u \in V - \{s\}\}$. It will then be sufficient to run the algorithm TREE-coloring, given in Section 3, on the balanced spanning tree T to get a solution for $(G, \mathbf{r}, s)$.

We recall that G is biconnected if it does not contain an articulation point, that is, a node whose removal along with its incident edges disconnects the graph [4].

Lemma 2. *Let G be a biconnected graph. If $r_u = 1$ for each $u \in V - \{s\}$, then there exists a balanced spanning tree of G rooted in v, for any $v \in V$.*

Proof. Let T be a spanning tree of G rooted in v. If T is not balanced, let $T_1, \cdots, T_m$ be the subtrees of v in T and w.l.o.g. suppose that $|T_1|$ is the largest. By Definition 4 we have $|T_1| > \sum_{i=2}^{m} |T_i| + 1$.

We modify T in order to get a balanced spanning tree of G. To this aim, we move one by one vertices of T_1 toward other subtrees. This is done by the moving step given in Fig. 3. Such a procedure, that decreases by 1 the size of the largest subtree, is iterated until the resulting spanning tree is balanced. □

Applying the TREE-coloring algorithm to the balanced spanning tree of G of Lemma 2, we get a CF–coloring for the instance $(T, \mathbf{r}, s)$. By Theorems 3 and 2, we have the following result.

Theorem 4. *Let $G = (V, E)$ be a biconnected graph, let $s \in V$ be the source, and $\mathbf{r} = \{r_u = 1 \mid u \in V - \{s\}\}$ be the set of requests. There exists a coloring algorithm that returns an optimal solution for the instance $(G, \mathbf{r}, s)$. The largest assigned color is $\mathcal{T}^*(G, \mathbf{r}, s) = |V| - 1$.*

```
Moving step: [Move one node from $T_1$ to some $T_i$, with $i \neq 1$]
Let $x$ be a node of maximum level in $T_1$ chosen among those such that there
exists $(x, y) \in E$ with $y \in V(T_i)$ for some $i \neq 1$.
1. while $x$ is an internal node:
         Let $x'$ be a descendant of $x$ in $T_1$ having a neighbor $z$ outside
          the subtree rooted in $x$
          [Since $x$ is not an articulation point, such a node must exist]
         Update $T_1$ by moving $x'$ as child of $z$;
          [This move strictly decreases the size of the subtree rooted in $x$]
2. [Here $x$ is a leaf]
    Update $T$ by moving $x$ from $T_1$ to $T_i$ as child of $y$.
```

Fig. 3. The Moving Step

5 Connected Graphs

Let $G = (V, E)$ be a connected graph. Fix a node s and assume $r_u = 1$ for each $u \in V - \{s\}$. We show how to obtain an optimal solution for G thus completing the proof of Theorem 1.

Our algorithm will be based on the individuation of the articulation points and the biconnected components (e.g maximal biconnected subgraphs) and bridges (or cut–edges) of the graph. It is well known that this can be achieved by a depth-first-search of the graph [4]. Denote by A the set of articulation points belonging to any biconnected component or bridge of G that also contains s. For each $a \in A$, let V_a denote the set of nodes $u \in V$ such that the removal of a together with its incident edges disconnects u and s. We can construct now a new graph G' obtained from G by compressing each set V_a, with $a \in A$, into the node a.

Definition 6. *The* compressed graph G' of G *is the weighted connected graph obtained from G by deleting, for each $a \in A$, all the nodes in V_a; the node a will be called a* supernode *of G' and is assigned weight $r'_a = |V_a \cup \{a\}|$. Each other node $u \in V(G') - A$ is assigned weight $r'_u = 1$.*

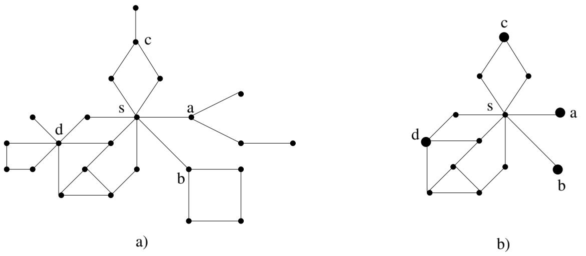


Fig. 4. a) A graph G with $A = \{a, b, c, d\}$. b) The compressed graph G' of G; a, b, c, d are supernode; $r_a = r_b = 4$, $r_c = 2$, $r_d = 5$, $r_u = 1$ for $u \notin A$.

In case we are able to construct a spanning tree of G' which is balanced with respect to the weights r'_v, then we can obtain an optimal solution for $(G, \mathbf{r}, s)$.

Lemma 3. *If G' admits a spanning tree rooted at s which is balanced with respect to the weights r'_v then one can obtain an optimal solution for $(G, \mathbf{r}, s)$.*

Unfortunately, a balanced spanning tree of G' does not always exists as shown in Fig. 5 b). On the positive side, we can efficiently obtain an optimal solution also in the absence of a balanced spanning tree. Indeed, we will give an optimal algorithm in case we do not have a balanced spanning tree of G'. To this aim we construct an "almost" balanced spanning tree of G'.

```
1. Start with a shortest path tree $S$ of $G'$ rooted at $s$.
2. Let the subtrees of $s$ in $S$ be $S_1, \ldots, S_m$ with $|S_1| \geq |S_i|$, for $i = 2, \ldots, m$,
   and let $v_1$ be the root of $S_1$.
3. **if** $(s, v_1)$ is a bridge **then** Output $S$.
4. **else** *[$S_1$ is in the same biconnected component of $G'$ as some $S_i$, $i \neq 1$]*
        - **while** $|S_1| > \sum_{j=2}^{m} |S_j| + 1$,
            update $S$ according to the moving procedure described in the
            proof of Lemma 2.
            *[Notice that a node is only moved inside one component of $G'$]*
        -*[Now $|S_1| \leq \sum_{j=2}^{m} |S_j| + 1$]* Output $S$.
```

If the tree S resulting from the above construction is balanced, then we can apply Lemma 3 and stop. Otherwise, we distinguish two cases according to whether edge (s, v_1) is a bridge or not.

Case 1: The edge (s, v_1) is a bridge
By construction of G', node v_1 is a supernode in G'. We substitute in S each supernode $a \in A \subseteq V(G')$ (having request equal to $|V_a \cup \{a\}|$ – see Definition 6) with a tree rooted at a and spanning all the nodes in $V_a \cup \{a\}$ (such nodes have now request 1 each). Applying TREE-coloring algorithm of Section 3 to the resulting spanning tree of G, by Theorem 3 we get a solution using $2|S_1| - 1$ colors. On the other hand, since (s, v_1) is a bridge (also in G), by ii) of Theorem 2 we get that $2|S_1| - 1$ colors are also necessary. Hence, the solution is optimal.

Case 2: The edge (s, v_1) is not a bridge
In this case the returned tree S is not balanced and it has $|S_1| \leq \sum_{j=2}^{m} |S_j| + 1$. Hence, for some $i \neq 1$

$$|S_i| > \sum_{j \neq i} |S_j| + 1.$$

Notice that (all nodes of) S_i must belong to the same biconnected component, say B, of G' as S_1. Indeed, any S_ℓ not belonging to B can not violate the balancing since $|S_\ell| \leq \sum_{j \neq \ell} |S_j| + 1$ before the construction and, during the construction, only nodes from S_1 are moved within B, and therefore not to S_ℓ. Hence both $|S_\ell|$ and $\sum_{j \neq \ell} |S_j| + 1$ remain identical at the end of the construction. Let w be the last moved node (from S_1 to S_i) and suppose we have moved w from having father $v \in V(S_1)$ to become a child of node u in S_i. Let us augment the tree T with the edge (v, w). We call the resulting graph a spanning tree of S *augmented* at w and denote it by S^A. For sake of simplicity we will keep addressing $S_1, \cdots S_m$ as subtrees of S^A even if S_1 is now augmented with w (which belongs now to both S_1 and S_i). The following facts are immediate.

Fact 1. *If the result of the above construction is a spanning tree augmented at w, then w is a supernode (i.e. has $r_w > 1$).*

Fact 2. *If S^A is a spanning tree augmented at w, then the path to w that uses only nodes in S_1 is the shortest path in G' from s to w.*

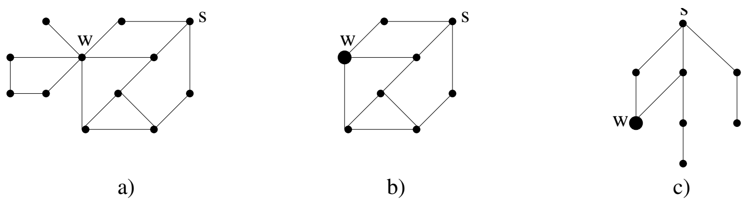


Fig. 5. a) Graph G; b) The compressed graph G' of G; c) A spanning tree S^A of G' augmented at w

Essentially in the augmented spanning tree, one makes node w be reachable using two node-disjoint paths in S^A; both paths will be used to serve the r'_w requests of w. We need now to decide how many of the requests of w will be routed through S_1 and how many should be routed through S_i. To this aim, we split the r'_w requests of w between S_1 and S_i as follows. Put

$$r^i_w = \lfloor \frac{\sum_{u \in V(T_1), u \neq w} r'_u + r'_w - \sum_{u \in V(T_i), u \neq w} r'_u}{2} \rfloor \qquad \text{and} \qquad r^1_w = r'_w - r^i_w.$$

We assign request r^1_w to w in S_1 and r^i_w requests to w in S_i (here we mean that we want r^1_w paths reach w passing through nodes in S_1); all the request of the other nodes remain unchanged.

Recalling that $|S_j|$ is the size (in terms of requests) of subtree S_j in S^A,

$$|S_j| = \begin{cases} \sum_{u \in V(S_j), u \neq w} r'_u + r^j_w & \text{if } j = 1 \text{ or } j = i, \\ \sum_{u \in V(S_j)} r'_u & \text{otherwise.} \end{cases}$$

It can be easily verified that S^A is balanced according to Definition 4.

A *balanced augmented spanning tree* T^A *of* G *rooted at* s can be easily derived from S^A by rooting in each supernode $a \in A \subseteq V(G')$ (see Definition 6) a spanning tree $T(a)$ of the subgraph of G induced by the vertices in $V_a \cup \{a\}$. In particular, by Fact 1, w is the root of tree $T(w)$ in the balanced augmented spanning tree T^A of G.

The Coloring Algorithm

We give now a coloring algorithm on the balanced augmented spanning tree T^A of G to obtain an optimal solution for $(G, \mathbf{r}, s)$.

For sake of simplicity we shall refer to $T_1, T_2, \cdots, T_m$ as subtrees of T^A even if two of them share the node w and subtree $T(w)$ rooted in it; w.l.o.g. in the following we assume them to be T_1 and T_2. As before, v_i denotes the root of T_i, for $i = 1, \ldots, m$. By the definition of T^A there are two paths from s to each node in $T(w)$, one going through T_1 and the other going through v_2 and nodes in T_2. Furthermore, denote by ℓ^j_w the length of the path from v_j to w in T_j, $j = 1, 2$; by Fact 2, we have $\ell^1_w \leq \ell^2_w$.

Definition 7. *Define*

$$T^1(w) = \begin{cases} \{\textit{the } r^1_w \textit{ nodes of higher level in } T(w)\} & \textit{if } \ell^2_w - \ell^1_w \textit{ is odd,} \\ \{\textit{the } r^1_w - 1 \textit{ nodes of higher level in } T(w)\} \cup \{w\} & \textit{otherwise,} \end{cases}$$

$$T^2(w) = T(w) - T^1(w).$$

For the description of the algorithm, we introduce the following terminology.

- *s serves $T^i(w)$, $i = 1, 2$*: s serves the tree $T^i(w)$ using the path from s going through T_i.
- *T_i is the largest subtree*: T_i is the subtree of T^A with the largest number of remaining requests.
- *s is active at step k*: s serves some subtree T_i during step k.

Since all the paths from the source s to the nodes in $T(w)$ have to go through node w, the order in which source s serves $T^1(w)$ and $T^2(w)$ is important to color these paths in a collision–free way. In the algorithm AUGMENTED-TREE-coloring we present below, the source s first serves all the nodes in $T^1(w)$ and then serves the nodes in $T^2(w)$. Hence we need to compute

$t =$ the step at which the source s can start to serve $T^2(w)$ after that it has served all the r^1_w nodes in $T^1(w)$.

To this aim, we assume to use the increasing coloring defined in Definition 5, to get a CF–coloring of the edges incident on w. Since s will serve alternatively $T^1(w)$ and same other T_i, $2r^1_w - 1$ steps are necessary to s to serve the r^1_w nodes in $T^1(w)$. By Definition 7, if $\ell^2_w - \ell^1_w$ is even then the last of the r^1_w paths to $T^1(w)$ stops at w; hence, the largest color assigned by the increasing coloring to the edges incident on w is $2r^1_w - 1 + \ell^1_w$. Otherwise, if $\ell^2_w - \ell^1_w$ is odd then the last of the r^1_w paths to nodes in $T^1(w)$ stops to a child of w; hence, the largest color assigned by the increasing coloring to the edges incident on w is $2r^1_w - 1 + \ell^1_w + 1$. If we define d as

$$d = \begin{cases} 2r^1_w + 1 + \ell^1_w - \ell^2_w & \text{if } \ell^2_w - \ell^1_w \text{ is odd,} \\ 2r^1_w + \ell^1_w - \ell^2_w & \text{otherwise,} \end{cases} \tag{2}$$

we have that if $d \geq 2$ then $t = d$; if $d \leq 0$ then the first useful step to start to serve nodes in $T^2(w)$ is 2. Hence, $t = \max\{2, d\}$ and by (2) t is even and $t \leq 2r^1_w$.

Finally, we denote by $\mathbf{p}_2$ the path in T_2 from v_2 to the last served node x in $T^2(w)$ and by ℓ_2 be the length of $\mathbf{p}_2$; notice that by Definition 7 node x is either w or a child of w depending on whether $w \in T^2(w)$ or not.

Definition 8. *Define* critical nodes of T_2 *the first* $\lfloor \frac{\ell_2}{2} \rfloor$ *nodes on the path* $\mathbf{p}_2$. *Subtree* T_2 *is called* locked *during the execution of the algorithm, if critical nodes of* T_2 *and nodes in* $T^2(w)$ *are the only remaining nodes to be satisfied in* T_2.

We can now formally describe the algorithm; it consists of a sequence of three phases, PHASE-1, PHASE-2, and PHASE-3, and is given in Fig. 6.

The following Lemmas allow to prove feasibility and optimality of the solution obtained by the AUGMENTED-TREE-coloring algorithm. This, together with the lower bound in Theorem 2, proves Theorem 1.

Lemma 4. *Let* $(\mathcal{P}, \hat{L})$ *be the solution for* $(G, \mathbf{r}, s)$ *returned by AUGMENTED-TREE-coloring algorithm. The increasing coloring* $\hat{L}$ *is a CF–coloring of* $\mathcal{P}$.

Lemma 5. *The largest color assigned at the end of algorithm AUGMENTED-TREE-coloring is equal to* $\max\{|V| - 1,\ d(s,w) + 2|T(w)| - 1\}$, *where* $d(s,w)$ *is the length of a shortest path connecting* s *and* w *in* G.

```
AUGMENTED-TREE-coloring ($T^A, \mathbf{r}, s$)
• Compute $t$ and set $\mathcal{P} = \emptyset$.
• [PHASE-1]
  for $k = 1$ to $t - 1$ do
  Step $k$
    if $k$ is odd then  $s$ serves $T^1(w)$.
    else
      if there exists a $T_i$ that is not satisfied, with $i \neq 1$ and $i \neq 2$ if $T_2$ is locked
        then $s$ serves the largest $T_i$, with $i \neq 1$ and with $i \neq 2$ if $T_2$ is locked;
        else $s$ remains idle
    if $s$ is not idle then
      Let $u$ be the served node, where $u$ is not a critical node of $T_2$ if $T_2$ is served.
      Color, using the $k$-increasing coloring $\hat{L}$, the obtained path $\mathbf{p}_u$,
      Put $\mathcal{P} = \mathcal{P} \cup \{\mathbf{p}_u\}$.
• [PHASE-2]
  for $k = t$ to $2r_w^1$ do
  Step $k$
     if $k$ is odd then  $s$ serves $T^1(w)$.
     else  $s$ serves the largest $T_i$, with $i \neq 1$.
     Let $u$ be the node which is served.
     Color, using the $k$-increasing coloring $\hat{L}$, the obtained path $\mathbf{p}_u$.
     Put $\mathcal{P} = \mathcal{P} \cup \{\mathbf{p}_u\}$.
• [PHASE-3]
  Set $g = 0$.
  while $k > 2r_w^1$ and $s$ has to serve do
  Step $k$
     if there exists a $T_i$, with $i \neq g$ that is not satisfied then
              - $s$ serves the largest subtree $T_i$, with $i \neq g$.
              - Let $u$ be the node which is served.
              - Color, using the $k$-increasing coloring $\hat{L}$, the obtained path $\mathbf{p}_u$.
              - Put $\mathcal{P} = \mathcal{P} \cup \{\mathbf{p}_u\}$ and $g = i$.
     else $s$ remains idle; set $g = 0$.
     k=k+1.
• return ($\mathcal{P}, \hat{L}$)
```

Fig. 6. The augmented tree coloring algorithm

6 Conclusions

In this paper, we have given an optimal solution to the collision-free path coloring problem in any graph. This translates into optimal time gathering algorithms for any type of topology of a sensor network. Our results improve on existing approximation algorithms. In particular, we have shown that if the topology of the network is biconnected then it is always possible to construct a particular rooted spanning tree, called balanced, that allows to obtain paths whose edges can be optimally colored. Our results refer to the gathering of one data packet from each node; as future work, an interesting problem is to obtain optimal solutions to the gathering problem when each node can hold more than one packet to be delivered to the sink.

The complexity of the gathering problem in the most general case in which some nodes can also have no packets to deliver is an open issue. We believe gathering in this general setting is an NP-complete problem; it would be interesting to prove it and, eventually, give a PTAS for the problem.

References

1. Akyildiz I.F., Su W., Sankarasubramaniam Y., Cayirci E. : Wireless sensor networks:a survey. Computer Networks, **38** (2002) 393–422.
2. Bermond J.-C., Galtier J., Klasing R., Morales N., Perennes S.: Hardness and approximation of gathering in static radio networks. Proceedings FAWN06 (2006).
3. Chong C-Y, Kumar S.P.: Sensor networks:Evolution, opportunities, and challenges. Proceedings of the IEEE ,**91 (8)** (2003) 1247-1256.
4. Cormen T.H., Leiserson C.E., Rivest R.L.: Introduction to algorithms. Mc Graw Hill 1990.
5. Dasgupta K., Kukreja M., Kalpakis K.: Topology-aware placement and role assignment for energy-efficient information gathering in sensor networks. Proceedings IEEE ISCC'03 (2003) 341-348.
6. Florens C., Franceschetti M.,McEliece R.J. Lower Bounds on Data Collection Time in Sensory Networks. IEEE Journal on Selected Areas in Communications, **22 (6)** (2004) 1110–1120.
7. Ho B., Prasanna V.K.: Constrained flow optimization with application to data gathering in sensor networks. Proceedings Proceedings of ALGOSENSORS 2004, LNCS **3121** (2004) 187-200.
8. Krishnamachari B., Estrin D., Wicker S.: Modeling data-centric routing in wireless sensor networks. Proceedings of IEEE INFOCOM 2002, (2002).
9. Kahn J.M.,Katz R.H., Pister K.S.J.: Mobile Networking for Smart Dust. Proceedings of ACM MobiCom 99, (1999).
10. Lindsey S., Raghavendra C., Sivalingam K.M.: Data gathering algorithms in sensor networks using energy metrics. IEEE Transactions on Parallel and Distributed Systems **13 (9)** (2002) 924-935.
11. Pelc A.: Broadcasting in radio networks . Handbook of Wireless Networks and Mobile Computing, I. Stojmenovic, Ed. John Wiley and Sons, Inc.,(2002) 509-528.
12. Shen C.,Srisathapornphat C., Jaikaeo C.: Sensor information networking architecture and applications. IEEE Personal Communications, (2001) 52-59.
13. Yu Y., Krishnamachari B., Prasanna V.: Energy-latency tradeoffs for data gathering in wireless sensor networks. Proceedings of IEEE INFOCOM 2004, (2004).
14. Zhu X., Tang B., Gupta H.: Delay efficient data gathering in sensor networks. Proceedings of MSN 2005, LNCS **3794** (2005) 380-389.

Magic Numbers in the State Hierarchy of Finite Automata[⋆]

Viliam Geffert

Department of Computer Science – P. J. Šafárik University
Jesenná 5 – 04001 Košice – Slovakia
geffert@upjs.sk

Abstract. A number d is magic for n, if there is no regular language for which an optimal nondeterministic finite state automaton (nfa) uses exactly n states, but for which the optimal deterministic finite state automaton (dfa) uses exactly d states. We show that, in the case of unary regular languages, the state hierarchy of dfa's, for the family of languages accepted by n-state nfa's, is not contiguous. There are some "holes" in the hierarchy, i.e., magic numbers in between values that are not magic. This solves, for automata with a single letter input alphabet, an open problem of existence of magic numbers [7].

As an additional bonus, we get a universal lower bound for the conversion of unary d-state dfa's into equivalent nfa's: nondeterminism does not reduce the number of states below $\log^2 d$, not even in the best case.

Keywords: Descriptional complexity, finite-state automata.

1 Introduction and Preliminaries

Automata theory is one of the oldest topics in theoretical computer science, and also a popular first step to study this field. In spite of that, some important problems are still open. The most famous problem is whether a two-way nondeterministic finite state automaton with n states can be converted into an equivalent two-way deterministic automaton using only a polynomial number of states [14]. (See also [4].)

At first glance, the situation is clear for one-way automata. By the classical subset construction [13], we know that a nondeterministic finite state automaton (nfa) with n states can be replaced by an equivalent deterministic finite state automaton (dfa) with d states, such that $n \leq d \leq 2^n$. In the worst case, $d = 2^n$. (See [10,12].) On the other hand, we also know languages for which nondeterminism does not help at all, that is, $d = n$. Thus, a natural question, raised for the first time by Iwama et al. [7], is the following:

> Is it possible, for a given number n, to find a number d satisfying $n \leq d \leq 2^n$, such that *no* optimal dfa using exactly d states can be simulated by *any* optimal nfa using exactly n states?

⋆ This work was supported by the Science and Technology Assistance Agency under contract APVT-20-004104, and by the Slovak Grant Agency for Science (VEGA) under contract "Combinatorial Structures and Complexity of Algorithms."

In [8], such numbers were named **magic numbers**, as numbers for which nondeterminism is especially weak. Adopting this terminology, we say that d is a **muggle number** for n, if it is not magic for n, i.e., if at least one optimal dfa using exactly d states can be simulated by an optimal nfa using exactly n states. A negative answer to the above question would show that all numbers between n and 2^n are muggle numbers, without leaving any "holes" in the hierarchy.

The above problem is easier to solve, if the size of the input alphabet grows in n [9]: For each n and d, such that $n \leq d \leq 2^n$, there exists an optimal n-state nfa for which the optimal dfa uses exactly d states. However, the size of the input alphabet for these automata is very large, namely, $2^{n-1}+1$. In the second part of [9], the input alphabet is reduced to $2n$. However, this is shown by a "non-constructive" argument, proving the mere existence without giving an explicit exhibition of witness languages. Finally, in [3], the complete state hierarchy is shown by a simpler "constructive" proof, displaying explicitly the witness automata and, at the same time, reducing the alphabet size to $n+2$.

In chronological order, the first work concerning this problem [7] was devoted to binary regular languages. It was shown, for each n, that values $d = 2^n - 2^k$ and $d = 2^n - 2^k - 1$, where $0 \leq k \leq n/2-2$, are not magic. Later, in [8], this has been extended for $d = 2^n - k$, where $5 \leq k \leq 2n-2$. In [9], new muggle numbers have been found at the opposite end, namely, for $n \leq d \leq 1+n\cdot(n+1)/2$. Finally, in [3], a superpolynomial number of muggle numbers has been presented, namely, each d satisfying $n \leq d \leq e^{\Theta(n^{1/3}\cdot\ln^{2/3} n)}$ is muggle. However, the completeness of the state hierarchy for the binary regular languages is an open problem.

In this paper, we shall focus our attention on the state hierarchy of *unary* regular languages, i.e., on automata with a single letter input alphabet. Unary (tally) languages play an important role in theoretical computer science, as languages with a very low information content. (See, e.g., [1,2,4,11].)

For unary regular languages, we present an almost exact approximation of $G_{\mathrm{max}}(n)$ and $G_{\mathrm{min}}(n)$, the largest and the smallest muggle numbers for n, respectively. Then we prove the existence of magic numbers between $G_{\mathrm{min}}(n)$ and $G_{\mathrm{max}}(n)$. Thus, in the unary case, the state hierarchy of dfa's, for the family of languages accepted by n-state nfa's, is not contiguous. We shall actually show that *most of the numbers* between $G_{\mathrm{min}}(n)$ and $G_{\mathrm{max}}(n)$ is magic. (A typical structure of the state hierarchy is shown in Fig. 2.)

In order to prove the existence of magic numbers for unary automata, we need to revise some of their properties first. In 1986, Chrobak [2] introduced a new normal form for unary nfa's, and used this normal form to show that the cost of eliminating nondeterminism in the unary case is at most $F(n)+O(n^2) \leq e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}}$ states, where $F(n)$ denotes the Landau's function. (For exact definition, see (1) below.) This is better than the standard subset construction.

We need to introduce a more refined version of the Chrobak normal form. This will reduce the cost of eliminating nondeterminism to $F(n-1) + (n^2-2)$ states. Such improvement seems to be marginal at first glance. However, in Sect. 4, we shall present an optimal nfa using exactly n states, such that its optimal deterministic counterpart uses exactly $F(n-1) + k_n$ states, for some

$k_n \in \{0,\ldots,n^2-2\}$. Thus, the new upper bound is almost equal to the actually existing optimum. A potential difference of at most n^2-2 states is negligible, compared with the growth rate of $F(n)$, which is $e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}}$, by (2) below.

Then we shall derive some properties of unary automata that are deterministic. Among others, an optimal dfa consisting of an initial segment of length s and a loop of length ℓ cannot be simulated by a dfa using a shorter initial segment or a loop of length ℓ' that factorizes into a "simpler" product of prime powers than does ℓ, even if the new machine uses far more states in total.

After that, using the conflict between the optimal dfa's and the revised Chrobak normal form for nfa's, we can prove the existence of magic numbers. As a simple consequence, we shall also get a new universal lower bound for the conversion of unary d-state dfa's into equivalent nfa's: nondeterminism does not reduce the number of states below $\ln^2 d$, not even in the best case.

We first briefly recall some basic definitions and notation on finite state automata. For more details, we refer the reader to [6].

Two automata are *equivalent*, if they accept the same language. An nfa (dfa) M is *optimal*, if no nfa (dfa, respectively) with a fewer number of states is equivalent to M.

For simplicity, a single-step transition from a state q to q' is presented as an edge $q \to q'$. A path reading a string 1^u from the input (thus, consisting of u consecutive edges) can be displayed in a more compact form $q \xrightarrow{1^u} q'$. Finally, $q \rightsquigarrow q'$ indicates reachability by a path of any length (including zero, for $q = q'$).

The factorization of integers will also be important in the subsequent considerations. For a more detailed exposition, the reader is referred to [5].

Let X be a finite multiset of positive integers, with possible repetition of elements. Then $\operatorname{lcm} X$ denotes the least common multiple of all elements in X and $\gcd X$ the greatest common divisor of these elements.

The Fundamental Theorem of Arithmetic states that each $\ell > 1$ can be uniquely *factorized* in the form $\ell = p_{i_1}^{\alpha_1} \cdot p_{i_2}^{\alpha_2} \cdot \ldots \cdot p_{i_e}^{\alpha_e}$, where $p_{i_1}^{\alpha_1}, p_{i_2}^{\alpha_2}, \ldots, p_{i_e}^{\alpha_e}$ are some prime powers. The set of prime powers used in the factorization of ℓ will be denoted by $\varphi(\ell) = \{p_{i_1}^{\alpha_1}, p_{i_2}^{\alpha_2}, \ldots, p_{i_e}^{\alpha_e}\}$. Clearly, $\ell = \prod_{p^\alpha \in \varphi(\ell)} p^\alpha$. We shall also need a *cost of factorization*, defined by

$$\Phi(\ell) = \textstyle\sum_{p^\alpha \in \varphi(\ell)} p^\alpha\,.$$

By definition, $\varphi(1) = \emptyset$, which gives $\Phi(1) = 0$.

Lemma 1. *For each $\ell \geq 1$, $\Phi(\ell) \leq \ell$. If, moreover, ℓ divides some ℓ', then $\Phi(\ell) \leq \Phi(\ell')$. For each $\ell_1, \ell_2, \ldots, \ell_m$, $\Phi(\operatorname{lcm}\{\ell_1, \ell_2, \ldots, \ell_m\}) \leq \sum_{i=1}^{m} \Phi(\ell_i)$.*

The following function will play a crucial role in our considerations. Let

$$F(n) = \max\{\operatorname{lcm}\{\ell_1, \ell_2, \ldots, \ell_m\} : \ell_1+\ell_2+\cdots+\ell_m = n\}\,. \tag{1}$$

This function, giving the largest least common multiple among all partitions of n, is known as Landau's function. The best known approximation of $F(n)$ is due

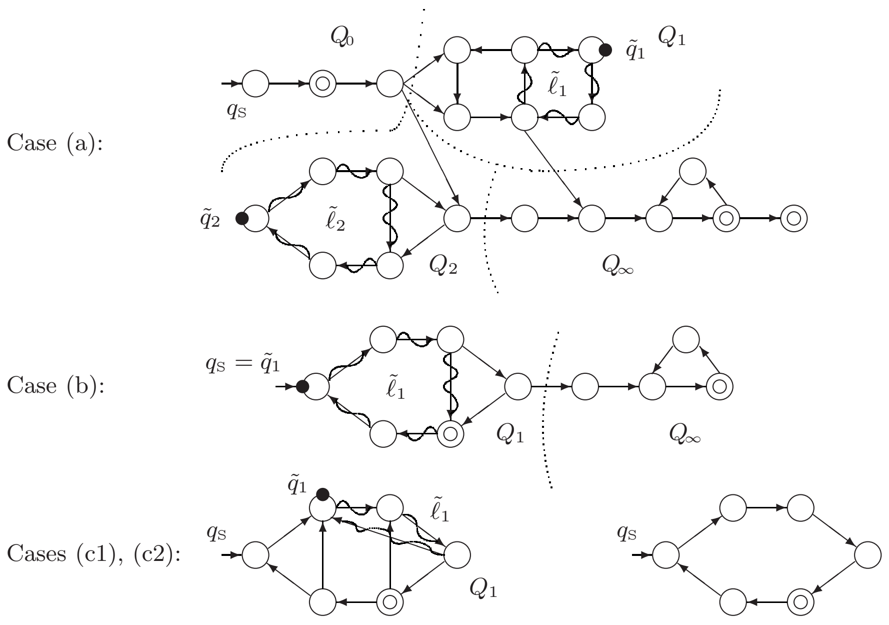


Fig. 1. Fixing cardinal loops and cardinal states. Partitioning of the state set into $Q_0 \cup Q_1 \cup \ldots \cup Q_m \cup Q_\infty$ is represented by dotted territorial boundaries, cardinal loops are marked by wavy lines twining around the edges, and cardinal states by filled bullets. (For details, see Def. 2.)

to Szalay [15]: $F(n) = e^{\sqrt{n\cdot(\ln n + \ln\ln n - 1 + (\ln\ln n - 2 + o(1))/\ln n)}}$. For our purposes, this approximation can be simplified as follows:

$$F(n) = e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}}\,. \tag{2}$$

2 Unary Nondeterministic Automata

Definition 2 (Cardinal loops and states). *Let M be a unary nfa using n states. Fix some loops in M, together with some states along these loops, as* cardinal*, in the following way. (There are several cases, illustrated by Fig. 1.)*

Case (a). *There is no loop $q_S \xrightarrow{1^\alpha} q_S$ beginning and ending in the initial state q_S, for no $\alpha > 0$. That is, q_S does not belong to any strongly connected component. Partition the state set into $Q = Q_0 \cup Q_1 \cup \ldots \cup Q_m \cup Q_\infty$ as follows:*

- Q_0 *contains all states that are reachable from q_S, but not reachable by computations passing through some loops. This implies that states in Q_0 do not belong to any strongly connected component, and that $q_S \in Q_0$.*
- Q_i, *for $i = 1, \ldots, m$, contains all states forming the ith strongly connected component in Q, reachable directly from Q_0. That is, if $q \in Q_i$, then (i) all states q' with paths $q \rightsquigarrow q' \rightsquigarrow q$ are included in Q_i, and (ii) there must exist a path $q_S \rightsquigarrow q$ consisting only of states in $Q_0 \cup Q_i$.*

Q_∞ *contains all remaining states, i.e., states either not reachable at all, or reachable only by computations passing through some states in* $\bigcup_{i=1}^{m} Q_i$.

Now, for $i = 1,\ldots,m$, *let* $\tilde{\ell}_i$ *denote the length of the shortest loop in* Q_i. *For each* Q_i, *fix one such loop (there may potentially be more than one), and also a state* $\tilde{q}_i \in Q_i$ *along this loop. The fixed loops of lengths* $\tilde{\ell}_1, \tilde{\ell}_2, \ldots, \tilde{\ell}_m$ *will be* cardinal loops, *the states* $\tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_m$ *will be* cardinal states. *(If* Q *does not contain any reachable strongly connected component,* $m = 0$.*) Since these loops are in pairwise disjoint components and there is no loop passing through* q_{S},

$$\tilde{\ell}_1+\tilde{\ell}_2+\cdots+\tilde{\ell}_m \leq n-1\,. \tag{3}$$

Case (b). *There exists a loop* $q_{\mathrm{S}} \xrightarrow{1^\alpha} q_{\mathrm{S}}$ *beginning and ending in the initial state* q_{S}, *for some* $1 \leq \alpha \leq n$–1. *Here we get a partition in the form* $Q = Q_1 \cup Q_\infty$, *that is,* $Q_0 = \emptyset$, $q_{\mathrm{S}} \in Q_1$, *and* $m = 1$. *Therefore, we fix some shortest loop beginning and ending in* q_{S} *as cardinal, of length* $\tilde{\ell}_1$, *and take* $\tilde{q}_1 = q_{\mathrm{S}}$ *as the only cardinal state. It is easy to see that (3) is satisfied again.*

Case (c). *There exists a loop beginning and ending in* q_{S}, *but the length of any such loop is at least* n. *Since the shortest loop does not repeat the same state twice, we have a path* $q_{\mathrm{S}} \xrightarrow{1^n} q_{\mathrm{S}}$, *visiting all states in* Q. *By enumerating all states in order in which they appear in the loop* $q_{\mathrm{S}} \xrightarrow{1^n} q_{\mathrm{S}}$, *we get*

$$q_{\mathrm{S}} = q_0 \rightarrow q_1 \rightarrow q_2 \quad \cdots \quad q_{n-2} \rightarrow q_{n-1} \rightarrow q_0\,. \tag{4}$$

Case (c1). *Besides the transitions displayed in (4), there exists at least one more edge in* M. *Such edge must be of the form* $q_e \leftarrow q_f$, *with* $1 \leq e \leq f \leq n-1$. *(Otherwise, we get either an edge already displayed in (4), or a loop* $q_{\mathrm{S}} \rightsquigarrow q_{\mathrm{S}}$ *shorter than* n. *Both cases lead to contradictions.)*

Now, let $\bar{e}$ *be the smallest* $e \geq 1$ *such that there exists a "backward" edge* $q_{\bar{e}} \leftarrow q_f$, *for some* $f \geq \bar{e}$. *Then let* $\bar{f}$ *be the smallest* $f \geq \bar{e}$ *with a backward edge* $q_{\bar{e}} \leftarrow q_{\bar{f}}$. *Finally, fix the loop* $q_{\bar{e}} \rightarrow q_{\bar{e}+1} \;\cdots\; q_{\bar{f}-1} \rightarrow q_{\bar{f}} \rightarrow q_{\bar{e}}$ *as cardinal, and fix* $\tilde{q}_1 = q_{\bar{e}}$ *as the only cardinal state. Clearly, the condition (3) is satisfied again.*

Case (c2). *The automaton* M *does not have any transitions except for those displayed in (4). In this case,* M *is already* deterministic. *We shall call such automaton a* trivial loop of length n, *and handle this special case separately.*

The following technical theorem serves as a tool for converting nondeterministic automata into a normal form.

Theorem 3. *Let* M *be a unary nfa with at most* $n > 1$ *states, different from the trivial loop of length* n, *with cardinal loops of lengths* $\tilde{\ell}_1, \tilde{\ell}_2, \ldots, \tilde{\ell}_m$ *and cardinal states* $\tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_m$, *as introduced by Def. 2. Then, for each* $u \geq n^2$–2, *the string* 1^u *is accepted by* M *if and only if*

(i) *there exists an* $i \in \{1,\ldots,m\}$ *(that is, also a strongly connected component* Q_i *with a cardinal loop* $\tilde{\ell}_i$ *and a cardinal state* $\tilde{q}_i$*),*

(ii) *there exists an* $r \in \{0,\ldots,\tilde{\ell}_i-1\}$ *(a number modulo* $\tilde{\ell}_i$*),*

(iii) *there exists a* $q' \in F$ *(a final state of* M*),*

(iv) *there exists an* $\alpha \leq n{-}1$, *with a computation path* $q_{\mathrm{S}} \xrightarrow{1^{\alpha}} \tilde{q}_i$,
(v) *there exists a* $\beta \leq n^2{-}n{-}1$, *with a computation path* $\tilde{q}_i \xrightarrow{1^{\beta}} q'$,
(vi) *such that* $(\alpha{+}\beta) \bmod \tilde{\ell}_i = r$,
(vii) *and* $u \bmod \tilde{\ell}_i = r$.

Now we can fix some "significant" parts of a nondeterministic computation.

Definition 4. *Let* M *be a unary nfa with at most* n *states, different from the trivial loop of length* n, *with cardinal loops* $\tilde{\ell}_1, \tilde{\ell}_2, \ldots, \tilde{\ell}_m$ *and cardinal states* $\tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_m$. *For each* $i \in \{1, \ldots, m\}$ *and each* $r \in \{0, \ldots, \tilde{\ell}_i{-}1\}$, *define*

- *a boolean predicate* $P_{i,r} = \mathit{true}/\mathit{false}$, *depending on whether*
  (iii) *there exists a* $q' \in F$,
  (iv) *there exists an* $\alpha \leq n{-}1$, *with a path* $q_{\mathrm{S}} \xrightarrow{1^{\alpha}} \tilde{q}_i$,
  (v) *there exists a* $\beta \leq n^2{-}n{-}1$, *with a path* $\tilde{q}_i \xrightarrow{1^{\beta}} q'$,
  (vi) *such that* $(\alpha{+}\beta) \bmod \tilde{\ell}_i = r$,
- *a set of indices* $R_i = \{r\colon 0 \leq r \leq \tilde{\ell}_i{-}1 \text{ and } P_{i,r} = \mathit{true}\}$,
- *a language* $L_{i,r} = \{1^u\colon u \geq n^2{-}2 \text{ and } u \bmod \tilde{\ell}_i = r\}$,
- *a language* $L_0 = \{1^u\colon u < n^2{-}2 \text{ and } u \in L(M)\}$.

Clearly, the truth of the predicates $P_{i,r}$ can be "precomputed" without knowing u, the length of the input. The same holds for the sets R_i as well. (This only requires to know the transition table of M, together with the allocation of cardinal loops and states.)

Theorem 5 (Chrobak normal form revised). *Let* M *be a unary nfa with at most* $n > 1$ *states, different from the trivial loop of length* n. *Then* M *can be replaced by an equivalent nfa* M' *consisting of an initial deterministic path of length* $\tilde{s} \leq n^2{-}2$, *and some* m *disjoint deterministic loops of lengths* $\tilde{\ell}_1, \ldots, \tilde{\ell}_m$, *with the total number of states in loops bounded by* $\tilde{\ell}_1{+}\cdots{+}\tilde{\ell}_m \leq n{-}1$. M' *makes a single nondeterministic decision (if any), after passing through the initial path, when it chooses one of the* m *loops (if* $m > 1$*).*

Proof. By Thm. 3, the string 1^u of length $u \geq n^2{-}2$ is in $L(M)$ if and only if it satisfies the statement of the items (i) – (vii). Using Def. 4, this holds if and only if there exists an $i \in \{1, \ldots, m\}$ and an $r \in R_i$, such that $1^u \in L_{i,r}$. But then $L(M) = L_0 \cup \bigcup_{i=1}^{m} \bigcup_{r \in R_i} L_{i,r}$.

It is easy to construct an M' for $L_0 \cup \bigcup_{i=1}^{m} \bigcup_{r \in R_i} L_{i,r}$. It consists of

- an initial segment, made up of some states $p_1, p_2, \ldots, p_{n^2-2}$, connected by edges $p_k \to p_{k+1}$, for $k = 1, \ldots, n^2{-}3$, with p_1 as the initial state,
- a separate loop of length $\tilde{\ell}_i$, for each $i \in \{1, \ldots, m\}$, made up of states $q_{i,0}, q_{i,1}, \ldots, q_{i,\tilde{\ell}_i-1}$, connected by $q_{i,k} \to q_{i,(k+1) \bmod \tilde{\ell}_i}$, for $k = 0, \ldots, \tilde{\ell}_i{-}1$,
- edges $p_{n^2-2} \to q_{i,0}$, for $i \in \{1, \ldots, m\}$, connecting the initial segment to each of the loops. This is the only nondeterministic decision, ever made.
- Finally, mark as accepting each state $q_{i,k}$ in the loop of length $\tilde{\ell}_i$ such that, for some $r \in R_i$, $(n^2{-}2{+}k) \bmod \tilde{\ell}_i = r$, and mark as accepting each p_k in the initial segment such that $1^{k-1} \in L(M)$. □

Theorem 6. *Let M be a unary nfa with at most $n>1$ states, different from the trivial loop of length n. Then M can be replaced by an equivalent dfa M'' consisting of an initial segment of length $\tilde{s} \leq n^2-2$, and a loop of length $\tilde{\ell}$ satisfying $\Phi(\tilde{\ell}) \leq n-1$.*

Proof. The deterministic automaton M'' is obtained by the standard subset construction [13], from the nfa M' constructed in Thm. 5. If we reduce the state set of M'' to the subset that is actually reachable from the initial state, M'' uses an initial segment of length n^2-2, together with a loop of length $\tilde{\ell} = \mathrm{lcm}\{\tilde{\ell}_1, \tilde{\ell}_2, \ldots, \tilde{\ell}_m\}$. Moreover, by Lem. 1, the length $\tilde{\ell}$ has a very low factorization cost, bounded by $\Phi(\tilde{\ell}) \leq \sum_{i=1}^{m} \Phi(\tilde{\ell}_i) \leq \sum_{i=1}^{m} \tilde{\ell}_i \leq n-1$. □

Theorem 7. *Let M be a unary nfa with at most $n>1$ states. Then M can be replaced by an equivalent dfa M'' using $d \leq F(n-1) + (n^2-2) \leq e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}}$ states.*

Proof. First, if M is different from the trivial loop of length n, M'' is obtained by the use of Thm. 6. This automaton uses $d = \tilde{\ell} + (n^2-2)$ states, with $\Phi(\tilde{\ell}) \leq n-1$. This allows us to express $\tilde{\ell}$ in the form $\tilde{\ell} = \mathrm{lcm}\{1, \ldots, 1, p_{i_1}^{\alpha_1}, \ldots, p_{i_e}^{\alpha_e}\}$, where "1" is repeated $(n-1) - \Phi(\tilde{\ell})$ times and $\{p_{i_1}^{\alpha_1}, \ldots, p_{i_e}^{\alpha_e}\}$ is the set of prime powers forming the factorization of $\tilde{\ell}$. The sum of these numbers is exactly $1+\cdots+1+p_{i_1}^{\alpha_1}+\cdots+p_{i_e}^{\alpha_e} = (n-1) - \Phi(\tilde{\ell}) + \Phi(\tilde{\ell}) = n-1$. But then $\tilde{\ell} \leq \max\{\mathrm{lcm}\{\ell_1, \ldots, \ell_f\} : \ell_1+\cdots+\ell_f = n-1\} = F(n-1)$. Thus, using (2), the number of states in M'' can be bounded by $d = \tilde{\ell} + (n^2-2) \leq F(n-1) + (n^2-2) \leq e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}}$.

If M turns out to be the trivial loop of length n, then M is already deterministic, so we take $M'' = M$, with $d = n$. □

3 Unary Deterministic Automata

It is obvious that the transition function of an optimal unary dfa is determined by two quantities, the length of the initial segment and the length of the subsequent loop. However, two dfa's may also differ in the distribution of their final states.

Theorem 8. *Let M_1,M_2 be two unary dfa's accepting the same language L, consisting of initial segments of lengths s_1,s_2 and loops of lengths ℓ_1,ℓ_2, respectively. Then L can also be accepted by a dfa M consisting of an initial segment of length $s = \min\{s_1, s_2\}$ and a loop of length $\ell = \gcd\{\ell_1, \ell_2\}$.*

We do not claim that M in the above theorem is optimal. Nevertheless, the theorem yields some consequences for automata that are optimal.

Theorem 9. *Let M be an optimal unary dfa consisting of an initial segment of length s and a loop of length ℓ. Then each dfa M', equivalent to M, must use an initial segment of length $s' \geq s$ and a loop of length ℓ' satisfying $\Phi(\ell') \geq \Phi(\ell)$.*

Proof. Let M and M' be two machines satisfying the assumptions of the theorem. By Thm. 8, we can replace M and M' by an equivalent dfa M'' with an initial segment of length $s'' = \min\{s, s'\}$ and a loop of length $\ell'' = \gcd\{\ell, \ell'\}$.

Suppose, for contradiction, that $\ell'' < \ell$. Then the total number of states in M'' can be bounded by $s''+\ell'' < s''+\ell = \min\{s, s'\}+\ell \leq s+\ell$. Thus, M'' uses fewer states than does M. But this is a contradiction, since M is optimal. Therefore, $\ell'' \geq \ell$. On the other hand, we have that $\ell'' = \gcd\{\ell, \ell'\}$ divides ℓ, and hence $\ell'' \leq \ell$. Summing up, $\ell'' = \ell$.

Second, $\ell'' = \gcd\{\ell, \ell'\}$ divides also ℓ' and hence, by Lem. 1, we get that $\Phi(\ell'') \leq \Phi(\ell')$. Therefore, $\Phi(\ell') \geq \Phi(\ell'') = \Phi(\ell)$.

Finally, if $s' < s$, then $s''+\ell'' = \min\{s, s'\}+\ell'' < s+\ell'' = s+\ell$, which again contradicts the fact that M is optimal. Therefore, $s' \geq s$. □

Theorem 10. *Let M be an optimal unary dfa consisting of an initial segment of length s and a loop of length ℓ, such that $s+\ell > n$, for some $n > 1$. If, moreover, either $s \geq n^2-1$ or $\Phi(\ell) \geq n$, then each nfa M', equivalent to M, must use more than n states.*

Proof. Suppose, for contradiction, that M can be replaced by an equivalent nfa M' with at most n states. If M' is different from the trivial loop of length n, we can use Thm. 6 to obtain an equivalent dfa M'', consisting of an initial segment of length $s'' \leq n^2-2$, and a loop of length ℓ'' satisfying $\Phi(\ell'') \leq n-1$.

But, by Thm. 9, the dfa M'', equivalent to M, must use the initial segment of length $s'' \geq s$ and the loop of length ℓ'' satisfying $\Phi(\ell'') \geq \Phi(\ell)$.

Summing up, we get $s \leq s'' \leq n^2-2$, and also $\Phi(\ell) \leq \Phi(\ell'') \leq n-1$. But this contradicts the assumption that either $s \geq n^2-1$ or $\Phi(\ell) \geq n$.

Finally, if M' is a trivial loop of length n, then M' is already deterministic, with $n < s+\ell$ states, which contradicts the assumption that M is optimal. □

4 Magic Numbers in the State Hierarchy

Theorem 11. *Let $G_{\max}(n)$ and $G_{\min}(n)$ denote, respectively, the largest and the smallest muggle numbers for $n > 1$. Then $G_{\max}(n) = F(n-1) + k_n$, for some $k_n \in \{0, \ldots, n^2-2\}$, which can be approximated by $G_{\max}(n) = e^{(1\pm o(1))\cdot\sqrt{n\cdot\ln n}}$, and $G_{\min}(n) = n$.*

Proof. For each n, consider the sequence of languages $L_0, L_1, \ldots, L_{n^2-1}$, where

$$L_k = \{1^{k+u} : u \bmod F(n-1) \neq 0\}, \quad \text{for } k = 0, \ldots, n^2-1.$$

The construction of a dfa M_k for L_k is straightforward. It consists of

- an initial segment $p_1 \to p_2 \to p_3 \ \ldots \ p_k \to q_0$, skipping the first k symbols, where p_1 is the initial state, and q_0 the first state of the subsequent loop,
- a loop $q_0 \to q_1 \to q_2 \ \ldots \ q_{F(n-1)-1} \to q_0$, counting modulo $F(n-1)$.
- Finally, all states in the loop, except for q_0, are marked as accepting.

It is easy to see that M_k is optimal, and uses exactly $F(n-1) + k$ states.

Now, let f_k denote the exact number of states used in an optimal nfa for L_k.

First, we shall show that $f_0 \leq n$. Let $F(n-1)$ factorize into $F(n-1) = p_{i_1}^{\alpha_1} \cdot p_{i_2}^{\alpha_2} \cdot \ldots \cdot p_{i_e}^{\alpha_e}$. Then $1^v \in L_0$ if and only if there exists some $j \in \{1, \ldots, e\}$

such that v is not divisible by $p_{i_j}^{\alpha_j}$. Therefore, L_0 can be accepted by an nfa M that nondeterministically chooses some j and then verifies if $v \bmod p_{i_j}^{\alpha_j} \neq 0$. Clearly, such automaton M uses $1+\sum_{j=1}^{e} p_{i_j}^{\alpha_j} = 1+\Phi(F(n-1))$ states. Using (1), $F(n-1) = \mathrm{lcm}\{\ell_1,\ldots,\ell_m\}$, for some $\ell_1,\ldots,\ell_m$ satisfying $\ell_1+\cdots+\ell_m = n-1$. But then, by Lem. 1, $1+\Phi(F(n-1)) \leq 1+\sum_{i=1}^{m}\Phi(\ell_i) \leq 1+\sum_{i=1}^{m}\ell_i = n$. We do not claim that M is optimal. For our purposes, it is sufficient to conclude that an optimal nfa for L_0 cannot use more states than does M, and hence $f_0 \leq n$.

Second, $f_{n^2-1} > n$. This follows from the fact that the optimal dfa M_{n^2-1} for L_{n^2-1}, described above, contains the initial segment of length $k = n^2-1$. But then, by Thm. 10, each nfa accepting L_{n^2-1} must use more than n states.

Third, $f_{k+1} \leq f_k + 1$, for each $k = 0,\ldots,n^2-2$. Let M'_k be an optimal nfa for L_k, with f_k states. To obtain an nfa M'' (not necessarily optimal) for L_{k+1}, we need only a new initial state q''_{s}, connected by a new edge $q''_{\mathrm{s}} \to q'_{\mathrm{s}}$ to the original initial state of M'_k. The rest is a direct simulation of M'_k. Clearly, for each v, M'' accepts $\mathtt{1}^{1+v}$ if and only if M'_k accepts $\mathtt{1}^v$. But an optimal nfa M'_{k+1} accepting L_{k+1} cannot use more states than does M'', and hence $f_{k+1} \leq f_k + 1$.

Summing up, we have an integer sequence $f_0, f_1, \ldots, f_{n^2-1}$, such that $f_0 \leq n$, $f_{n^2-1} > n$, and $f_{k+1} \leq f_k + 1$. Such sequence must contain an element equal to n, that is, $f_{k'} = n$, for some $k' \in \{0,\ldots,n^2-2\}$. But then $L_{k'}$ is a language for which the optimal nfa $M'_{k'}$ uses exactly $f_{k'} = n$ states and the optimal dfa $M_{k'}$ exactly $F(n-1) + k'$ states.

Therefore, $F(n-1)+k'$ is a muggle number for n. But then the largest muggle number is at least $G_{\max}(n) \geq F(n-1)$. On the other hand, by Thm. 7, each unary nfa with n states can be replaced by an equivalent dfa, not necessarily optimal, using at most $F(n-1) + (n^2-2)$ states. This gives $G_{\max}(n) \leq F(n-1) + (n^2-2)$. Using (2), we then get $G_{\max}(n) = e^{(1\pm o(1))\cdot\sqrt{n\cdot\ln n}}$.

For completeness, $G_{\min}(n) = n$. The language $L = \{\mathtt{1}^u \colon u \bmod n = 0\}$ requires exactly n states, both in deterministic and nondeterministic case. □

We are now going to prove the existence of *nontrivial* magic numbers, between $G_{\min}(n)$ and $G_{\max}(n)$, i.e., "holes" in the state hierarchy.

Definition 12 (Darkly magic numbers). *A number d is* darkly magic *for n, if, for each positive integer $\ell \in \{d-n^2+2,\ldots,d-1,d\}$, $\Phi(\ell) \geq n$.*

Theorem 13. *Let d be a darkly magic number for $n > 1$. Then, for each optimal unary dfa M using exactly d states, an optimal nfa M', equivalent to M, must use more than n states.*

Proof. Using Lem. 1, we get $\Phi(d) \leq d \leq n-1$, for each $d \leq n-1$. If $d = n$, then $\Phi(d-1) \leq n-1$. Thus, using an $\ell' \in \{d-1, d\}$, we get $\Phi(\ell') \leq n-1$, if $d \leq n$. But this contradicts the assumption that d is darkly magic for n. Therefore, $d > n$.

Now, let M be an optimal unary dfa, using d states. M consists of an initial segment of length $s \geq 0$ and a loop of length $\ell \geq 1$, such that $s+\ell = d > n$. Further, if the initial segment is of length $s \leq n^2-2$, the loop length is at least $\ell = d-s \geq d - n^2 + 2$. But then $\Phi(\ell) \geq n$, since d is darkly magic for n.

Summing up, $s+\ell > n$, and either $s \geq n^2-1$ or $\Phi(\ell) \geq n$. But then, by Thm. 10, each nfa M', equivalent to M, must use more than n states. □

Thus, to show the existence of nontrivial magic numbers, it is sufficient to prove the existence of nontrivial darkly magic numbers.

Lemma 14. *Let $F_{\#}(n)$ denote the number of different values d satisfying $\Phi(d) \leq n$. Then $F_{\#}(n) \leq e^{(1+o(1))\cdot 2\sqrt{\ln 2}\cdot\sqrt{n}}$.*

Lemma 15. *There are at most $F_{\#}(n-1)\cdot(n^2-1) \leq e^{(1+o(1))\cdot 2\sqrt{\ln 2}\cdot\sqrt{n}}$ different numbers that are not darkly magic for $n>1$. Consequently, each set containing more than $F_{\#}(n-1)\cdot(n^2-1)$ positive integers contains at least one number that is darkly magic for n.*

Proof. If d is not darkly magic for n, it can be expressed in the form $d = s+\ell$, for some ℓ satisfying $\Phi(\ell) \leq n-1$, and some $s \in \{0,\ldots,n^2-2\}$. But, by Lem. 14, there are only $F_{\#}(n-1)$ different numbers ℓ with factorization cost $\Phi(\ell) \leq n-1$, and n^2-1 different values of s. □

Theorem 16. *Let $M_{\min}(n)$,$M_{\max}(n)$ denote the smallest and the largest nontrivial magic numbers, and $D_{\min}(n)$,$D_{\max}(n)$ the smallest and the largest nontrivial darkly magic numbers for n, respectively. Except for some finitely many n's, such numbers do exist, and $G_{\min}(n) < M_{\min}(n) \leq D_{\min}(n) < D_{\max}(n) \leq M_{\max}(n) < G_{\max}(n)$. Moreover, $D_{\min}(n) \leq e^{(1+o(1))\cdot 2\sqrt{\ln 2}\cdot\sqrt{n}}$, and $D_{\max}(n) = e^{(1\pm o(1))\cdot\sqrt{n\cdot\ln n}}$.*

Proof. Consider the set $X = \{1,\ldots,F_{\#}(n-1)\cdot(n^2-1)+1\}$. By Lem. 15, this set contains at least one number that is darkly magic for n. Let $D_{\min}(n)$ be the smallest darkly magic number in X. First, $D_{\min}(n) > n$, since a darkly magic number must be larger than n, as shown in the proof of Thm. 13. Second, a darkly magic number larger than n is, by Thm. 13, also a magic number larger than n. Therefore, there must exist $M_{\min}(n) \leq D_{\min}(n)$, the smallest magic number larger than n. Using $G_{\min}(n) = n$, shown by Thm. 11, we thus get $G_{\min}(n) < M_{\min}(n) \leq D_{\min}(n) \leq F_{\#}(n-1)\cdot(n^2-1)+1$.

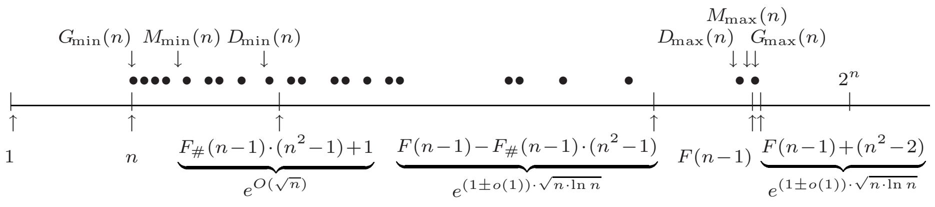
By the same reasoning for $Y = \{G_{\max}(n) - F_{\#}(n-1)\cdot(n^2-1),\ldots,G_{\max}(n)\}$, we obtain $G_{\max}(n) - F_{\#}(n-1)\cdot(n^2-1) \leq D_{\max}(n) \leq M_{\max}(n) < G_{\max}(n)$.

Finally, using the growth rates that were presented in Lem. 15 and Thm. 11, we obtain that $F_{\#}(n-1)\cdot(n^2-1)+1 < G_{\max}(n) - F_{\#}(n-1)\cdot(n^2-1)$. □

Actually, most of the numbers between $G_{\min}(n)$ and $G_{\max}(n)$ is magic. The structure of the state hierarchy is shown in Fig. 2.

Corollary 17. *Let $G_{\#}(n)$ denote the number of muggle numbers, and $M_{\#}(n)$ the number of nontrivial magic numbers for n. Then $G_{\#}(n) \leq e^{(1+o(1))\cdot 2\sqrt{\ln 2}\cdot\sqrt{n}}$, and $M_{\#}(n) = e^{(1\pm o(1))\cdot\sqrt{n\cdot\ln n}}$. Consequently, $\lim_{n\to\infty} G_{\#}(n)/M_{\#}(n) = 0$.*

Corollary 18. *(a) For each $d > 1$, no optimal unary dfa using d states can be simulated by an nfa using fewer than $(1-o(1))/2\cdot\ln^2 d/\ln\ln d \geq \Omega(\ln^2 d/\ln\ln d)$ states. (b) For infinitely many d's, no optimal unary dfa using d states can be simulated by an nfa using fewer than $(1-o(1))/(4\ln 2)\cdot\ln^2 d \not< o(\ln^2 d)$ states.*



$G_{\min}(n)$
$M_{\min}(n)$
$D_{\min}(n)$
$M_{\max}(n)$
$D_{\max}(n)$
$G_{\max}(n)$
2^n
1
n
$F_{\#}(n-1)\cdot(n^2-1)+1$
$e^{O(\sqrt{n})}$
$F(n-1)-F_{\#}(n-1)\cdot(n^2-1)$
$e^{(1\pm o(1))\cdot\sqrt{n\cdot\ln n}}$
$F(n-1)$
$F(n-1)+(n^2-2)$
$e^{(1\pm o(1))\cdot\sqrt{n\cdot\ln n}}$

Fig. 2. An example of a typical distribution of muggle and magic numbers for n. Here the "x-axis" grows in d, the number of states in dfa's. The filled bullets along this axis represent muggle numbers, while the "white space" surrounding the bullets represents magic numbers.

Proof. (a) By Thm. 7, no optimal dfa with d states can be simulated by an n-state nfa, if $e^{(1+o(1))\cdot\sqrt{n\cdot\ln n}} < d$. Thus, $n \geq (1-o(1))/2 \cdot \ln^2 d/\ln\ln d$.

(b) Consider the sequence $D_{\min}(n_0), D_{\min}(n_0+1), D_{\min}(n_0+2), \ldots$, starting from a sufficiently large n_0. Since $D_{\min}(n) > n$, by Thms. 16 and 11, this sequence must contain infinitely many different integers. By Thm. 16, we also have, for each $n \geq n_0$, that $D_{\min}(n) \leq e^{(1+o(1))\cdot 2\sqrt{\ln 2}\cdot\sqrt{n}}$. This gives that $n \geq (1-o(1))/(4\ln 2)\cdot \ln^2(D_{\min}(n))$. Moreover, $D_{\min}(n)$ is darkly magic for n and hence, by Thm. 13, no optimal dfa using $D_{\min}(n)$ states can be simulated by an nfa using fewer than $n+1$ states. □

5 Concluding Remarks

We have shown that, in the unary case, the state hierarchy of deterministic automata, for the family of languages accepted by nondeterministic automata using n states, is not contiguous. There are some "holes" in the hierarchy, i.e., magic numbers between the smallest muggle number $G_{\min}(n) = n$ and the largest muggle number $G_{\max}(n) = e^{(1\pm o(1))\cdot\sqrt{n\cdot\ln n}}$. In addition, if $G_{\#}(n)$ is the total number of different muggle numbers for n, and $M_{\#}(n)$ the number of nontrivial magic numbers, then $\lim_{n\to\infty} G_{\#}(n)/M_{\#}(n) = 0$. It is easy to see that there must exist two muggle numbers d_1, d_2, with $d_2 - d_1 \geq e^{\Omega(\sqrt{n\cdot\ln n})}$, such that all values between d_1 and d_2 are magic. This illustrates that some holes in the hierarchy, consisting of consecutive magic numbers, are quite spacious.

As a by-product of the conversion presented in Sect. 2, a superpolynomial gap between the size of unary nfa's and dfa's can be obtained only by machines *not* having the initial state in any strongly connected component.

We also have a new *universal* lower bound for the conversion of unary dfa's into equivalent nfa's. Clearly, using $L = \{1^u : u \bmod d = 0\}$, we get a dfa with d states that cannot be simulated by an nfa with a smaller number of states. This gives an "existential" lower bound $\Omega(d)$, showing that nondeterminism does not help in the worst case at all. On the other hand, Cor. 18 shows that, for unary languages, nondeterminism *never* reduces the number of states below $\Omega(\ln^2 d/\ln\ln d)$, for no d, and no optimal unary dfa M using d states, not even in the best case. Moreover, there are infinitely many “critical” values of d, for which nondeterminism does not reduce the number of states to $o(\ln^2 d)$.

We do not have a sufficient upper bound for $M_{\min}(n)$, the smallest nontrivial magic number for n, except for $M_{\min}(n) \leq e^{O(\sqrt{n})}$, given by Thm. 16. A better upper bound for the growth rate of $M_{\min}(n)$ would result in a better universal lower bound in Cor. 18. But the most important problem in this field is the completeness of the state hierarchy for the regular languages over the binary alphabet, or any other fixed input alphabet.

References

1. Bertoni, A., Mereghetti, C., Pighizzini, G.: An optimal lower bound for nonregular languages. Inform. Process. Lett. **50** (1994) 289–92. (Corrigendum: ibid. **52** (1994) p. 339)
2. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. **47** (1986) 149–58. (Corrigendum: ibid. **302** (2003) 497–98)
3. Geffert, V.: (Non)determinism and the size of one-way finite automata. In Proc. Descr. Compl. Formal Syst. (2005) 23–37. (IFIP & Univ. Milano)
4. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoret. Comput. Sci. **295** (2003) 189–203
5. Hardy, G., Wright, E.: *An Introduction to the Theory of Numbers*. Oxford University Press, 5th edit., 1979
6. Hopcroft, J., Motwani, R., Ullman, J.: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001
7. Iwama, K., Kambayashi, Y., Takaki, K.: Tight bounds on the number of states of DFA's that are equivalent to n-state NFA's. Theoret. Comput. Sci. **237** (2000) 485–94
8. Iwama, K., Matsuura, A., Paterson, M.: A family of NFA's which need $2^n - \alpha$ deterministic states. Theoret. Comput. Sci. **301** (2003) 451–62
9. Jirásková, G.: Note on minimal finite automata. In Proc. Math. Found. Comput. Sci., Lect. Notes Comput. Sci. **2136** (2001) 421–31
10. Lupanov, O. B.: Uber den Vergleich zweier Typen endlicher Quellen. Probleme der Kybernetik **6** (1966) 329–35. (Akademie-Verlag, Berlin, in German)
11. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. **30** (2001) 1976–92
12. Moore, F.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans. Comput. **C-20** (1971) 1211–14
13. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Develop. **3** (1959) 114–25
14. Sakoda, W., Sipser, M.: Nondeterminism and the size of two-way finite automata. In Proc. ACM Symp. Theory of Comput. (1978) 275–86
15. Szalay, M.: On the maximal order in S_n and S_n^*. Acta Arith. **37** (1980) 321–31

Online Single Machine Batch Scheduling[⋆]

Beat Gfeller[1], Leon Peeters[1], Birgitta Weber[2,⋆⋆], and Peter Widmayer[1]

[1] Institute of Theoretical Computer Science, ETH Zurich, Switzerland
{gfeller, peetersl, widmayer}@inf.ethz.ch
[2] Department of Computer Science, University of Liverpool, UK
weberb@csc.liv.ac.uk

Abstract. We are concerned with the problem of safely storing a history of actions that happen rapidly in real time, such as in "buy" and "sell" orders in stock exchange trading. This leads to a single-family scheduling problem with batching on a single machine, with a setup time and job release times, under batch availability. We investigate the objective of minimizing the total flow time in an online setting. On the positive side, we propose a 2-competitive algorithm for the case of identical job processing times, and we prove a lower bound that comes close. With general processing times, our lower bound shows that online algorithms are inevitably bad in the worst case.

Keywords: Scheduling, batching, online analysis, competitive ratio.

1 Introduction

The study in this paper is motivated by the real world problem of saving a log of actions in a high throughput environment. Many actions are to be carried out in rapid succession, and in case of a system failure the log can identify which actions have been carried out before the failure and which have not. Keeping such a log can be existential for a business, for example when logging the trading data in a stock brokerage company.

Logging takes place on disk and is carried out by a storage system that accepts write requests. When a process wants its data to be logged, it sends a log request with the log data to the storage system and waits for the acknowledgement that the writing of the log data has been completed. For the log requests that arrive over time at the storage system, there is only one decision the system is free to make: What subset of the requested, but not yet written log data should be written to disk in a single large write operation to make the whole system as efficient as possible? After the chosen large write operation is complete, the system instantaneously sends acknowledgements to all processes whose requests have been satisfied.

⋆ We gratefully acknowledge the support of the Swiss SBF under contract no. C05.0047 within COST-295 (DYNAMO) of the European Union.

⋆⋆ This research was carried out while the author was affiliated with the Institute of Theoretical Computer Science, ETH Zurich, Switzerland.

The difficulty in the above question stems from the following. First, log data come in all sizes, as the number of bits or blocks to be stored varies. Second, writing several log data in a single shot is faster than writing each of them individually, due to the disk hardware constraints. Third, a process requesting a write has to wait for the acknowledgement of the completion of the large write operation before it can continue. Based on an experimental evaluation of writing data to disk, we assume the writing time for a number of data blocks to be linear in that number, plus an additive constant for the disk write setup time. Our objective is to minimize the sum over all requests of the times between the request's arrival and its acknowledgement.

We ignore the details of a failure and its recovery here, assuming that failures are rare and recovery is quick, and not worrying about (the potential loss of) unsatisfied write requests.

1.1 Single Machine Scheduling with Batching

Viewing the storage system as a machine, the log requests as jobs, and the write operations as batches, this problem falls into the realm of scheduling with batching [for an overview, see Potts and Kovalyov, 2000]. More precisely, in the usual batch scheduling taxonomy we deal with a *family scheduling problem* with batching on a single machine (with one family). The machine processes the jobs consecutively, since the log data are stored consecutively in time on the disk (as opposed to simultaneously), and each batch of jobs requires a constant (disk write) setup time. As all log requests in a single write are simultaneously acknowledged at the write completion time, the machine operates with *batch availability*, meaning that each job in the batch completes only when the full batch is completed.

In more formal terms, we model the storage system as a single machine, and the log requests as a set of jobs $J = \{1, \ldots, n\}$. The arrival times of the log requests at the storage system then correspond to job release times $r_j, j \in \{1, \ldots, n\}$. Further, each job $j \in J$ has a processing time p_j on the machine, representing the block size of the log request. The grouping of the log requests into write operations is modeled by the batching of the jobs into a partition $\sigma = \{\sigma_1, \ldots, \sigma_k\}$ of the jobs $\{1, \ldots, n\}$, where σ_u represents the jobs in the u-th batch, k is the total number of batches, and we refer to $|\sigma_u|$ as the *size* of batch u. Unless stated otherwise, we assume that the batch size is not limited. We denote the starting time of batch σ_u by T_u, with $r_j \leq T_u$ for all $j \in \sigma_u$. Starting at T_u, the batch requires the constant disk setup time s, and further a total processing time $\sum_{j \in \sigma_u} p_j$, for writing the logs on the disk. Thus, each batch σ_u requires a total *batch processing time* $P_u = s + \sum_{j \in \sigma_u} p_j$. The consecutive execution of batches on the single machine translates into $T_u + P_u \leq T_{u+1}$ for $u = 1, \ldots, k-1$. Because of batch availability, each job $j \in \sigma_u$ completes at time $C_j = T_u + P_u$, and takes a flow time $F_j = C_j - r_j$ to be processed. This job flow time basically consists of two components: first a *waiting time* $T_u - r_j \geq 0$ that the job waits before batch σ_u starts, followed by the batch processing time P_u. Finally, as mentioned above, our objective is to minimize the total job flow time $\mathcal{F} = \sum_{j=1}^{n} F_j$.

We refer to this scheduling problem as the BatchFlow problem. In the standard scheduling classification scheme, the BatchFlow problem is written as $1|r_j, s_f = s, F = 1|\sum F_j$, where the part $s_f = s, F = 1$ refers to the fact that the jobs belong to a single family with a fixed batch setup time [see Potts and Kovalyov, 2000]. As a special case, we also consider the problem variant with *identical processing times* $p_j = p$.

1.2 Online Algorithms for a Single Machine with Batching

In this paper, we consider the *online* version of the single machine scheduling problem with batching. The jobs arrive over time, and any algorithm can base its batching decisions at a given time instant only on the jobs that have arrived so far. We study the online problem in the *non-preemptive clairvoyant* setting: No information about a job is known until it arrives, but once a job j has arrived, both its release time r_j (that is, arrival time) and processing time p_j are known. A batch that has started processing cannot be stopped before completion.

We consider *deterministic* online algorithms, and assume without loss of generality that the algorithms have a particular structure, as described in the following.

Since preemption is not allowed, no new batch can be started as long as the machine is busy. Further, it only makes sense for an online algorithm to revise a decision when new information becomes available, that is, when a new job arrives. Therefore, we consider online algorithms that only take a decision at the completion time of a batch σ_u, or when a new job j arrives and the machine is idle. We refer to these two events as triggering events. In either case, the algorithm bases its decision on the jobs $\{1, \ldots, j\}$ that have arrived so far, and on the batches $\sigma_1, \ldots, \sigma_u$ created so far. Note that the set $\mathcal{P}$ of currently pending jobs can be deduced from this information.

In case of a triggering event, an online algorithm $\mathcal{A}$ takes the following two decisions. First, it tentatively chooses the next batch $\sigma_{\mathcal{A}}$ to be executed on the machine. However, it does not execute the batch $\sigma_{\mathcal{A}}$ immediately. Rather, the algorithm's second decision defines a *delay time* $\Delta_{\mathcal{A}}$ by which it delays the execution of $\sigma_{\mathcal{A}}$, and waits for a triggering event to occur in the meantime. If $\Delta_{\mathcal{A}}$ time has elapsed, and no triggering event has happened, then the algorithm starts the batch $\sigma_{\mathcal{A}}$ on the machine (by definition, the machine is idle in this case). If, however, a triggering event occurs during the delay time, then the algorithm newly chooses $\sigma_{\mathcal{A}}$ and $\Delta_{\mathcal{A}}$. Thus, an online algorithm $\mathcal{A}$ is completely specified by how it chooses $\sigma_{\mathcal{A}}$ and $\Delta_{\mathcal{A}}$.

For a given problem instance I, let $\mathcal{F}_{\text{OPT}}(I)$ be the total flow time of an optimal solution, and $\mathcal{F}_{\mathcal{A}}(I)$ the total flow time of the solution obtained from some online algorithm $\mathcal{A}$ for the same problem. We are interested in the *competitive ratio* $\frac{\mathcal{F}_{\mathcal{A}}}{\mathcal{F}_{\text{OPT}}}$ of an algorithm $\mathcal{A}$. Recall that an online algorithm $\mathcal{A}$ is *c-competitive* if there is a constant α such that for all finite instances I, $\mathcal{F}_{\mathcal{A}}(I) \le c \cdot \mathcal{F}_{\text{OPT}}(I) + \alpha$. When this condition holds also for $\alpha = 0$, we say that $\mathcal{A}$ is *strictly c-competitive*.

1.3 Related Work

Scheduling with batching has been extensively studied in the offline setting [see Potts and Kovalyov, 2000]. The offline version of our BatchFlow problem is NP-complete in general, as we prove in the full version of this paper [Gfeller et al., 2006]. There, we also show that the problem becomes efficiently solvable if the jobs have to be processed in release order, or when all jobs have identical processing times. Note that the offline BatchFlow problem is equivalent to the problem $1|r_j, s_f = s, F = 1|\sum C_j$, since $\sum_{j \in J} r_j$ is a constant that we cannot influence.

Most previous work on online scheduling with batching focuses on the so-called *burn-in* model [see Lee et al., 1992], where the processing time of a batch is equal to the maximum processing time among all jobs in the batch. An exception is Divakaran and Saks [2001], who consider the problem $1|r_j, s_f|\max F_j$ with sequence-independent setup times and several job families under *job availability*, where the processing of a job completes as soon as its processing time has elapsed. They present an $O(1)$-competitive online algorithm for that problem. One typically distinguishes the *bounded* model, where the size of a batch can be at most B[1], from the *unbounded* model. For the burn-in model, Chen et al. [2004] consider the problem $1|r_j|\sum w_j C_j$, and present a 10/3-competitive online algorithm for unbounded batch size, as well as a $4 + \epsilon$-competitive online algorithm for bounded batch size.

The online problem $1|r_j|C_{\max}$ of minimizing the makespan in the burn-in model has been considered in several studies. Independently, Deng et al. [2003] and Zhang et al. [2001] gave a $(\sqrt{5}+1)/2$ lower bound for the competitive ratio, and both gave the same online algorithm for the unbounded batch size model which matches the lower bound. Poon and Yu [2005a] present a different online algorithm with the same competitiveness, and describe a parameterized online algorithm which contains their own and the previous solution as special cases. The same authors give a class of 2-competitive online algorithms for bounded batch size, and a 7/4-competitive algorithm for bounded batch size $B = 2$ in Poon and Yu [2005b].

In addition to the objective of minimizing the total flow time, some applications may require a guaranteed limit for the maximum flow time of any job in the BatchFlow problem, as this ensures that no job has to wait for an unbounded time. In Gfeller et al. [2006], however, we show that online algorithms have severe difficulties in dealing with such a constraint on the maximum flow time.

1.4 Contribution of the Paper

To our knowledge, we are the first to consider the online batch scheduling problem with the objective of minimizing the total flow time $\sum F_j$ under batch availability.

We propose the online algorithm Greedy as a solution to the online BatchFlow problem. For the special case of identical processing times p, we show that

[1] This bound is also called *capacity* by some authors.

Greedy is strictly 2-competitive, using the fact that its makespan is optimal up to an additive constant. Moreover, we present two lower bounds, $1+\frac{1}{1+\frac{\max(p,s)}{\min(p,s)}}$ and $1+\frac{1}{1+2\frac{p}{s}}$ for this problem variant, and hence show that Greedy is not far from optimal for this variant.

For the general online BatchFlow problem, we then give an $\frac{n}{2}-\epsilon$ lower bound for the competitive ratio, and show that any online algorithm which avoids unnecessary idle time, including Greedy, is strictly n-competitive, which matches the order of the lower bound.

The remainder of the paper is organized as follows. We first analyze the online BatchFlow problem for jobs with identical processing times $p_i = p$. For this case, we present the 2-competitive Greedy algorithm in Section 2, and derive two lower bounds in terms of p and s for any online algorithm in Section 3. Next, Section 4 discusses bounds for any online algorithm for the case of general processing times, and Section 5 concludes the paper.

2 The Greedy Batching Algorithm for Identical Processing Times

In this section, we consider the restricted case where all jobs have identical processing times $p_i = p$. This case is relevant in applications such as ours, where records of fixed length are to be logged.

We now define the Greedy algorithm, which always starts a batch consisting of all currently pending jobs as soon as the machine becomes idle. Formally, Greedy chooses $\Delta_{\mathcal{A}} = 0$, and sets $\sigma_{\mathcal{A}}$ equal to the set of currently pending jobs $\mathcal{P}$. First, we focus on the makespan of Greedy. The following theorem shows that, if Greedy needs time t to finish a set of batches $\{\sigma_1, \ldots, \sigma_u\}$, then no other algorithm can complete the same jobs before time $t - s$. Thus, Greedy is 1-competitive for minimizing the makespan, with an additive constant $\alpha = s$.

Theorem 1. *For a given problem instance of the online* BatchFlow *problem with identical processing times, let $\sigma = \sigma_1, \ldots, \sigma_k$ with batch starting times $T_1, \ldots, T_k$ be the* Greedy *solution, and let $\sigma' = \sigma'_1, \ldots, \sigma'_m$ with $T'_1, \ldots, T'_m$ be some other solution* Any *for the same instance. For any batch $\sigma_u \in \sigma$ completing at time t, it holds that any batch $\sigma'_v \in \sigma'$ satisfying the condition*

$$\sum_{i=1}^{v} |\sigma'_i| \geq \sum_{i=1}^{u} |\sigma_i| \tag{1}$$

completes at time $t' \geq t - s$.

Proof. For a given batch $\sigma_u \in \sigma$, consider the first batch $\sigma'_v \in \sigma'$ for which (1) holds. Such a batch exists because $\sum_{i=1}^{m} |\sigma'_i| = n$ and of course $\sum_{i=1}^{u} |\sigma_i| \leq n$. We assume that the Greedy batches $\sigma_1, \ldots, \sigma_u$ are executed without any idle time in between. Indeed, if such an idle time occurs, Greedy must have processed all jobs which have arrived so far, and the idle time ends exactly at the release

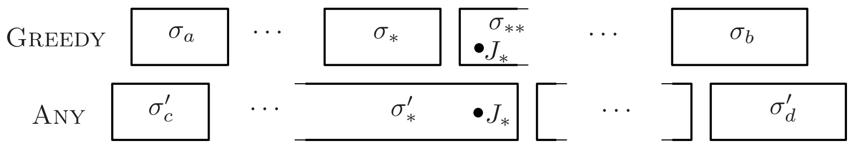


Fig. 1. Illustration of Lemma 1

time of the next job. Of course, Any cannot start processing this job earlier than Greedy does. Hence, ignoring all jobs before such an idle time can only affect the comparison in favor of Any.

First, we consider the case $u \geq v + 1$, where Greedy uses at least one batch more than Any. We require the following lemma, which is illustrated in Figure 1 and proven in the full version of this paper [Gfeller et al., 2006].

Lemma 1. *Consider a sequence $\sigma_a, \ldots, \sigma_b$ of u* Greedy *batches, and a sequence $\sigma'_c, \ldots, \sigma'_d$ of v* Any *batches, such that the* Any *sequence contains all the jobs in the* Greedy *sequence, and possibly additional jobs. If $u \geq v + 1$, then there exists a batch σ'_* among* Any*'s batches that contains both at least one entire* Greedy *batch, and at least one following job J_* from the next* Greedy *batch.*

Apply Lemma 1 to $\sigma_1, \ldots, \sigma_u$ and $\sigma'_1, \ldots, \sigma'_v$, and let σ'_* be the *last* Any batch containing a full Greedy batch followed by at least one job. Choose σ_* such that it is the last full Greedy batch in σ'_* that is followed by some job J_* in σ'_*. When J_* arrives, Greedy is already processing batch σ_* (or has just finished), because otherwise J_* would be part of σ_*. On the other hand, Any cannot start processing batch σ'_* before J_* arrives. So, it must hold that $T_{\sigma_*} \leq T_{\sigma'_*}$. Let σ_{**} be the Greedy batch following σ_*. Note that Lemma 1 (in contraposition) can be applied also to the sequences $\sigma_{**}, \ldots, \sigma_u$ and $\sigma'_*, \ldots, \sigma'_v$. Since in these two sequences no Any batch contains an entire Greedy batch followed by another job, we have $|\{\sigma_{**}, \ldots, \sigma_u\}| \leq |\{\sigma'_*, \ldots, \sigma'_v\}|$. Defining z as the number of batches in $\{\sigma_*, \ldots, \sigma_u\}$, and z' as the number of batches in $\{\sigma'_*, \ldots, \sigma'_v\}$, it holds $z \leq z' + 1$. Putting all of the above together, we obtain:

$$t - t' \leq T_{\sigma_*} + p \cdot (|\sigma_*| + \ldots + |\sigma_u|) + zs - T_{\sigma'_*} - p \cdot (|\sigma'_*| + \ldots + |\sigma'_v|) - z's \leq s$$

Finally, we consider the remaining case $u \leq v$. Since Greedy has no idle time, it completes σ_u exactly at time $t = p \cdot \sum_{i=1}^{u} |\sigma_i| + us$. Any finishes σ'_v at time $t' \geq p \cdot \sum_{i=1}^{v} |\sigma'_i| + vs$ or later. So, using (1), we obtain that $t' - t \geq (v - u)s$, proving the theorem for any $u \leq v + 1$, and for $u \leq v$ in particular. □

From this theorem, we obtain the following lemma.

Lemma 2. *In the online* BatchFlow *problem with identical processing times, consider any batch σ_u of the* Greedy *solution, with starting time T_u. Let σ' be the first batch of the optimal solution* Opt *that contains some job in σ_u. The earliest time that* Opt *can finish processing the m jobs in $\sigma_u \cap \sigma'$ is $T_u + mp$.*

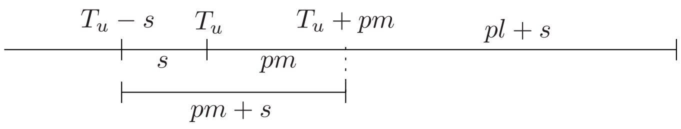


Fig. 2. Important time instants in GREEDY's competitiveness proof

Proof. Observe that, if we deleted all jobs in $\sigma_u \backslash \sigma'$ from the problem instance, then GREEDY would start processing exactly the m jobs in $\sigma_u \cap \sigma'$ in one batch at time T_u, and finish at $T_u + mp + s$. Now, if OPT were to finish these m jobs before $T_u + mp$, then there would exist a solution with makespan more than s smaller than GREEDY's makespan. This is a contradiction to Theorem 1. □

Note that the proofs of Theorem 1 and Lemma 2 can easily be adapted to incorporate non-identical processing times. We proved them for identical processing times here, since they serve as ingredients for the main theorem below, which only applies to identical processing times.

Now follows an observation concerning the optimal offline solution for a special case, which we need afterwards to compare online algorithms against the best possible solution.

Observation 1. *The total job flow time for optimally processing n jobs with identical processing times $p_j = p$ and with identical release times is at least*

$$\mathcal{F}_n \geq \frac{1}{2}pn^2 + sn.$$

Proof. Assume w.l.o.g. that all $r_j = 0$. The first job will have completion time at least $s + p$. The second job will finish no earlier than $s + 2p$, which can be achieved if the first two jobs are batched together. Generally, the ith job can finish no earlier than $s + ip$, which would be achieved by batching the first i jobs together. This shows that $\mathcal{F}_n \geq \sum_{i=1}^{n}(s+ip) = \frac{1}{2}pn(n+1)+sn \geq \frac{1}{2}pn^2+sn$. □

Theorem 2. *The* GREEDY *algorithm is strictly 2-competitive for the online* BATCHFLOW *problem with identical processing times.*

Proof. Figure 2 shows all the relevant time instants for the proof. As in Lemma 2, consider any batch σ_u of the GREEDY solution, with starting time T_u, and let σ' be the first batch of the optimal solution OPT that contains some jobs in σ_u. Below, we compare the total accumulated flow time before and after time T_u for the jobs in σ_u, for both GREEDY and OPT.

Lemma 2 implies that no job in σ_u can complete before T_u in OPT. Thus, up until time T_u, the jobs in σ_u have accumulated a total flow time of $\mathcal{F}^{\leq T_u}(\sigma_u) := \sum_{j\in\sigma_u}(T_u - r_j)$ in both GREEDY and OPT.

Let $\mathcal{F}^{\geq T_u}_{\text{GREEDY}}(\sigma_u)$ denote the total flow time for the jobs in σ_u after time T_u in the GREEDY solution, and $\mathcal{F}^{\geq T_u}_{\text{OPT}}(\sigma_u)$ the same quantity for the OPT solution. Further, we let $\mathcal{F}_{\text{OPT}}(\sigma_u) = \mathcal{F}^{\geq T_u}_{\text{OPT}}(\sigma_u) + \mathcal{F}^{\leq T_u}(\sigma_u)$ and $\mathcal{F}_{\text{GREEDY}}(\sigma_u) =$

$\mathcal{F}^{\geq T_u}_{\text{Greedy}}(\sigma_u) + \mathcal{F}^{\leq T_u}(\sigma_u)$ be the total flow time for the jobs in σ_u in Opt and Greedy, respectively.

Now, if σ' starts at $T_u - s$ or earlier, then all jobs in σ' must arrive at $T_u - s$ or earlier. Therefore, up until time T_u, the m jobs in $\sigma_u \cap \sigma'$ already yield an accumulated total flow time $\mathcal{F}^{\leq T_u}(\sigma_u) \geq ms$ in this case. After time T_u, the Greedy solution further accumulates a total flow time $\mathcal{F}^{\geq T_u}_{\text{Greedy}}(\sigma_u) = P_u = l(lp + s)$ for the l jobs in σ_u. As σ' finishes at or after T_u, and all jobs in σ_u must have arrived by T_u, the total flow time of Opt for the jobs in σ_u after time T_u is

$$\mathcal{F}^{\geq T_u}_{\text{Opt}}(\sigma_u) \geq \overbrace{m(pm)}^{\text{Lemma 2}} + \overbrace{(l-m)(pm)}^{\text{wait for } \sigma' \text{ to complete}} + \overbrace{\frac{1}{2}p(l-m)^2 + (l-m)s}^{\text{Observation 1}} . \quad (2)$$

Therefore, we have

$$\begin{aligned} 2 \cdot \mathcal{F}_{\text{Opt}}(\sigma_u) &\geq pl^2 + pm^2 + 2s(l-m) + 2\mathcal{F}^{\leq T_u}(\sigma_u) \\ &\geq pl^2 + sl + \mathcal{F}^{\leq T_u}(\sigma_u) = \mathcal{F}_{\text{Greedy}}(\sigma_u). \quad (3) \end{aligned}$$

Next, we consider the case in which σ' starts after $T_u - s$, say at starting time $T_u - s + \tau$, for $\tau > 0$. We still have $\mathcal{F}^{\geq T_u}_{\text{Greedy}}(\sigma_u) = l(lp + s)$ for Greedy. In this case, however, we obtain $\mathcal{F}^{\leq T_u}(\sigma_u) \geq m(s - \tau)$, and further an additive term $l\tau$ in the bound (2) for $\mathcal{F}^{\geq T_u}_{\text{Opt}}(\sigma_u)$. The additive term $l\tau$ for $\mathcal{F}^{\geq T_u}_{\text{Opt}}(\sigma_u)$ cancels out against the extra term $-m\tau$ for $\mathcal{F}^{\leq T_u}(\sigma_u)$ in the inequalities (3), so the bound (3) also applies in this case.

Since $\frac{\mathcal{F}_{\text{Greedy}}(\sigma_u)}{\mathcal{F}_{\text{Opt}}(\sigma_u)} \leq 2$ holds for any batch σ_u of the Greedy solution, the theorem follows. □

3 Lower Bounds for Identical Processing Times

Below, we derive two lower bounds for any algorithm for the online BatchFlow problem, again with identical processing times. These bounds show that no online algorithm can be much better than Greedy for this setting.

Theorem 3. *No online algorithm for the online* BatchFlow *problem with identical processing times can have a competitive ratio lower than*

$$1 + \frac{1}{1 + \frac{\max(p,s)}{\min(p,s)}}.$$

Proof. Let $\mathcal{A}$ be any online algorithm with finite delay time $\Delta_{\mathcal{A}}$ for $\mathcal{P} = \{1\}$. The adversary chooses release times $r_1 = 0$, $r_2 = \Delta_{\mathcal{A}}$, and $r_j = \Delta_{\mathcal{A}} + p(j-1) + s(j-2)$ for $j \in \{3, \ldots, n\}$, as depicted in Figure 3. Observe that an offline solution can avoid any waiting time for $n - 2$ jobs: If job 1 and job 2 are processed together, the first batch has finished just when job 3 arrives, so if job 3 is processed

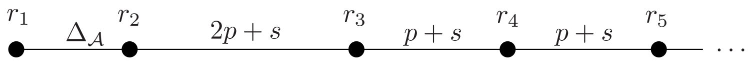


Fig. 3. The lower bound construction

immediately, it will be finished just when job 4 arrives, and so on until job n. Thus,

$$\mathcal{F}_{\mathrm{OPT}} \leq \overbrace{\Delta_{\mathcal{A}}}^{\text{job 1 waits}} + 2(2p+s) + (n-2)\cdot(p+s) = n(p+s) + \Delta_{\mathcal{A}} + 2p.$$

For bounding the flow time of $\mathcal{A}$'s solution, we examine for each job j the earliest possible completion time that $\mathcal{A}$ can achieve.

By construction of the example, $\mathcal{A}$ cannot batch job 2 together with job 1, and starts processing job 1 at time $\Delta_{\mathcal{A}}$, which completes at $C_1 = \Delta_{\mathcal{A}} + p + s$. Hence, job 2 cannot start processing before C_1, and thus $C_2 \geq \Delta_{\mathcal{A}} + 2p + 2s$. In the full version of this paper [Gfeller et al., 2006], we show that for each $j \in \{3, \dots, n\}$, it holds $C_j \geq \Delta_{\mathcal{A}} + pj + s(j-1) + \min(p,s)$. Adding $\sum_{j=1}^{n} F_j = \sum_{j=1}^{n} C_j - r_j$, we get

$$\mathcal{F}_{\mathcal{A}} \geq \overbrace{\Delta_{\mathcal{A}} + p + s}^{\text{job 1}} + \overbrace{2p + 2s}^{\text{job 2}} + \sum_{j=3}^{n} (p + s + \min(p,s))$$

$$= \Delta_{\mathcal{A}} + np + p + ns + s + (n-2)\min(p,s)$$

The competitive ratio can now be bounded as

$$\frac{\mathcal{F}_{\mathcal{A}}}{\mathcal{F}_{\mathrm{OPT}}} \geq \frac{\Delta_{\mathcal{A}} + np + p + ns + s + (n-2)\min(p,s)}{n(p+s) + \Delta_{\mathcal{A}} + 2p}$$

$$\rightarrow \frac{p + s + \min(p,s)}{p+s} = 1 + \frac{1}{1 + \frac{\max(p,s)}{\min(p,s)}} \quad \text{for } n \to \infty. \Box$$

Using a similar construction, one can show the following lower bound. For a proof, we refer to the full version of this paper [Gfeller et al., 2006].

Theorem 4. *No online algorithm for the online* BatchFlow *problem with identical processing times can have a competitive ratio lower than*

$$1 + \frac{1}{1 + 2\frac{p}{s}}.$$

4 Bounds on the Competitive Ratio for General Processing Times

We now consider the online BatchFlow problem with general processing times, where different jobs can have different processing times. Note that in this setting, it may be beneficial not to process jobs in the order they arrive, that is, to reorder jobs.

The following theorem shows that no online algorithm can have a good worst case performance for the online BatchFlow problem with general processing times, if reordering of jobs is allowed.

Theorem 5. *For the online* BatchFlow *problem, any online algorithm has competitive ratio at least $\frac{n}{2} - \epsilon$ for any $\epsilon > 0$.*

Proof. Let $\mathcal{A}$ be any online algorithm. Consider an instance of n jobs, where the first job 1 has processing time p, and all other jobs $2, \ldots, n$ have processing time 1, and arrive immediately after $\mathcal{A}$ starts processing job 1 (after having delayed for $\Delta_{\mathcal{A}}$ time units). As each of the jobs $2, \ldots, n$ has to wait for job 1 to finish, the total flow time for $\mathcal{A}$ is $\mathcal{F}_{\mathcal{A}} \geq \Delta_{\mathcal{A}} + n(p+s) + 1/2(n-1)^2 + (n-1)s$, using the lower bound from Observation 1 for optimally processing $(n-1)$ jobs of equal processing time arriving at the same time.

For Opt, consider the solution that first processes jobs $2, \ldots, n$ in one batch, and after that processes job 1: $\mathcal{F}_{\text{OPT}} \leq \Delta_{\mathcal{A}} + n((n-1) + s) + p + s$.

If $\Delta_{\mathcal{A}} > p + s$, then our bound for $\mathcal{F}_{\mathcal{A}}$ increases, but we can decrease the bound for $\mathcal{F}_{\text{OPT}}$ because Opt can complete job 1 even before the other jobs arrive, and then process all other jobs in one batch. Therefore, we assume in the following that $\Delta_{\mathcal{A}} \leq p + s$. We thus have

$$\mathcal{F}_{\mathcal{A}} \geq np + \frac{1}{2}n^2 + 2ns - n - s + \frac{1}{2} \quad \text{and} \quad \mathcal{F}_{\text{OPT}} \leq 2p + 2s + n^2 - n + ns.$$

It is easily verified that

$$\left(\frac{n}{2} - \epsilon\right) \cdot \mathcal{F}_{\text{OPT}} \leq \mathcal{F}_{\mathcal{A}} \quad \text{if we choose} \quad p \geq \frac{1}{4\epsilon}\left(n^3 + n^2 s + 2n + 2s + 2\epsilon n\right).$$

□

Theorem 5 shows that for the general setting, there is no online algorithm with a sub-linear competitive ratio. However, we will see in the following that all so-called non-waiting algorithms, a class to which the Greedy algorithm belongs, are strictly n-competitive, i.e., are at most a factor 2 away from the lower bound. We call an online algorithm *non-waiting* if it never produces a solution in which there is idle time while some jobs are pending.

Theorem 6. *Any non-waiting online algorithm for the online* BatchFlow *problem is strictly n-competitive.*

Proof. Let $\mathcal{A}$ be any non-waiting online algorithm. Consider any job i, and let σ_u be the batch which contains job i. Furthermore, let J' be the set of all jobs not contained in σ_u. The flow time of job i is $F_i = C_i - r_i = T_u + P_u - r_i$. The longest possible interval during which σ_u needs to wait (i.e., the machine is busy) in a non-waiting algorithm's solution is $s|J'| + \sum_{j \in J'} p_j$. So, for a non-waiting algorithm, $T_u - r_i \leq s(n-1) + \sum\limits_{j \in J'} p_j$. Hence,

$$F_i \leq P_u + s(n-1) + \sum_{j \in J'} p_j \leq sn + \sum_{j=1}^{n} p_j .$$

Thus, the total flow time for $\mathcal{A}$'s solution is

$$\mathcal{F}_{\mathcal{A}} = \sum_{i=1}^{n} F_i \leq n^2 s + n \cdot \sum_{j=1}^{n} p_j .$$

We now turn to the optimal solution OPT. Clearly, each job i has flow time $F'_i \geq p_i + s$, as the batch processing time is inevitable. Thus, the flow time of OPT is at least

$$\mathcal{F}_{\text{OPT}} = \sum_{i=1}^{n} F'_i \geq ns + \sum_{j=1}^{n} p_j .$$

Comparing the total flow times $\mathcal{F}_{\mathcal{A}}$ and $\mathcal{F}_{\text{OPT}}$ completes the proof. □

Observe that this upper bound proof does not make use of the fact that the reordering of jobs is allowed. Thus, requiring jobs to be processed in arrival order does not affect the validity of Theorem 6. Note that this is not true for Theorem 5.

We remark that the online non-preemptive scheduling problem with release times is a special case of the online BATCHFLOW problem. Thus, the $\Theta(n)$ upper bound for the former problem, mentioned by Epstein and van Stee [2003], is implied by our Theorem 6.

5 Conclusion and Open Problems

We considered the online BATCHFLOW problem, which is written as the family scheduling with batching problem $1|r_j, s_f = s, F = 1| \sum F_j$ in the standard scheduling classification scheme. For the online BATCHFLOW problem with identical processing times, we presented a 2-competitive greedy algorithm, as well as two lower bounds for any online algorithm. We also derived bounds for the general online BATCHFLOW problem. As summarized in Table 1, upper and lower bound are of the same order in the case of the general online BATCHFLOW problem (with arbitrary p_j). In the case of identical processing times ($p_j = p$ for all j), bounds are only matching if $s \gg p$, as then $1 + \frac{1}{1+2\frac{p}{s}}$ is close to 2. Further, we showed that no online algorithm can guarantee to compute a feasible solution for the online BATCHFLOW problem with a maximum processing time constraint whenever there is a feasible offline solution.

Table 1. Results for the online BATCHFLOW problem

online BATCHFLOW problem	lower bound	upper bound
general p_j	$\frac{n}{2} - \epsilon$	n
identical $p_j = p$	$1 + \frac{1}{1+2\frac{p}{s}}, \quad 1 + \frac{1}{1+\frac{\max(p,s)}{\min(p,s)}}$	2

Even though our greedy algorithm is nearly worst-case optimal for identical processing times, and not too far from optimal for the general online BATCH-FLOW problem, one might expect that online algorithms which wait at least some short time may be superior. An average case analysis for the online BATCHFLOW problem, assuming random release times, might lead to further insights in this question. Along with an average case analysis, one could consider randomized online algorithms.

Further, it could be interesting to consider special problem cases, for example, when the ratio between the maximum processing time and the setup time is bounded, or when the ratio between maximum and minimum processing time is bounded.

References

B. Chen, X. Deng, and W. Zang. On-line scheduling a batch processing system to minimize total weighted job completion time. *Journal of Combinatorial Optimization*, 8:85–95, 2004.

X. Deng, C.K. Poon, and Y. Zhang. Approximation algorithms in batch processing. *Journal of Combinatorial Optimization*, 7:247–257, 2003.

S. Divakaran and M. Saks. Online scheduling with release times and set-ups. Technical Report 2001-50, DIMACS, 2001.

L. Epstein and R. van Stee. Lower bounds for on-line single-machine scheduling. *Theoretical Computer Science*, 299(1-3):439–450, 2003.

B. Gfeller, L. Peeters, B. Weber, and P. Widmayer. Single machine batch scheduling with release times. Technical Report 514, ETH Zurich, 2006. Available at http://www.inf.ethz.ch/research/disstechreps/techreports.

C.Y. Lee, R. Uzsoy, and L.A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775, 1992.

C.K. Poon and W. Yu. A flexible on-line scheduling algorithm for batch machine with infinite capacity. *Annals of Operations Research*, 133:175–181, 2005a.

C.K. Poon and W. Yu. On-line scheduling algorithms for a batch machine with finite capacity. *Journal of Combinatorial Optimization*, 9:167–186, 2005b.

C. Potts and M. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

G. Zhang, X. Cai, and C.K. Wong. On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics*, 48:241–258, 2001.

Machines that Can Output Empty Words

Christian Glaßer and Stephen Travers[*]

Theoretische Informatik
Julius-Maximilians Universität Würzburg, Germany
{glasser, travers}@informatik.uni-wuerzburg.de

Abstract. We propose the e-model for leaf languages which generalizes the known balanced and unbalanced concepts. Inspired by the neutral behavior of rejecting paths of NP machines, we allow transducers to output empty words. The paper explains several advantages of the new model. A central aspect is that it allows us to prove strong gap theorems: For any class $\mathcal{C}$ that is definable in the e-model, either coUP $\subseteq \mathcal{C}$ or $\mathcal{C} \subseteq$ NP. For the existing models, gap theorems, where they exist at all, only identify gaps for the definability by *regular* languages. We prove gaps for the general case, i.e., for the definability by *arbitrary* languages. We obtain such general gaps for NP, coNP, 1NP, and co1NP. For the regular case we prove further gap theorems for Σ_2^{P}, Π_2^{P}, and Δ_2^{P}. These are the first gap theorems for Δ_2^{P}.

1 Introduction

Bovet, Crescenzi, and Silvestri [5] and Vereshchagin [20] independently introduced leaf languages. This concept allows a uniform definition of many interesting complexity classes like NP and PSPACE. The advantage of such an approach is that it allows to prove general theorems in a concise way. For example, Glaßer et al. [10] recently showed that if $\mathcal{C}$ is a class that is balanced-leaf-language definable by a regular language, then all many-one complete problems of $\mathcal{C}$ are polynomial-time many-one autoreducible. This general theorem answered several open questions, since classes like NP, PSPACE, and the levels of the PH are definable in this way.

Moreover, leaf languages allow concise oracle constructions. The background is the BCSV-theorem [5,20] that connects polylog-time reducibility (plt-reducibility) with the robust inclusion of two complexity classes (i.e., the inclusion with respect to all oracles). This connection reduces oracle constructions to their combinatorial core. In particular, neither do we have to care about the detailed stagewise construction of the oracle, nor do we have to describe the particular coding of the single stages.

In this paper we offer a useful generalization of the known leaf-language concepts. Despite of its broader definition, the new concept is convenient and has the nice features we appreciate with traditional leaf languages. It even combines

[*] Supported by the Konrad-Adenauer-Stiftung.

certain advantages of single known concepts. We summarize the benefit of the new notion:

1. works with balanced computation trees
2. admits a BCSV-theorem [5,20]
3. establishes a tight connection between the polynomial-time hierachy and the Straubing-Thérien hierarchy (the quantifier-alternation hierarchy of the logic FO[<] on words)

The new e-model of leaf languages is inspired by the observation that rejecting paths of nondeterministic computations act as *neutral* elements. In this sense we allow nondeterministic transducers not only to output single letters, but also to output the empty word ε which is the neutral element of Σ^*. More precisely, we consider nondeterministic polynomial-time-bounded Turing machines M such that on every input, every computation path stops and outputs an element from $\Sigma \cup \{\varepsilon\}$. Let $M(x)$ denote the computation tree on input x, and define $\beta_M(x)$ as the concatenation of all outputs of $M(x)$. For any language B, let $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$ (the e-class of B) be the class of languages L such that there exists a nondeterministic polynomial-time-bounded Turing machine M as above such that for all x,

$$x \in L \iff \beta_M(x) \in B.$$

If we demand that M never outputs ε, then this defines $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(B)$ (the u-class of B). If we demand that M is balanced and never outputs ε, then this defines $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(B)$ (the b-class of B). (M is *balanced* if there exists a polynomial-time computable function that on input (x, n) computes the n-th path of $M(x)$.) The notions e-class, u-class, and b-class are extended from a single language B to a class of languages $\mathcal{C}$ in the standard way: $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(\mathcal{C})$ (the e-class of $\mathcal{C}$) is the union of all $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$ where $B \in \mathcal{C}$. For a survey on the leaf-language approach we refer to Wagner [21].

It is immediately clear that the u-model and the b-model are restrictions of the e-model.

$$\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{b}}(B) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{u}}(B) \subseteq \mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$$

Moreover, it is intuitively clear that the presence of the neutral element ε gives the class $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$ some inherent nondeterministic power which makes $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$ seemingly bigger than P. We will discuss this issue and we will identify UP $\cap$ coUP as a lower bound (we obtain stronger bounds if we restrict to regular languages B). The advantage of the e-model over the u-model is its simplicity: In the e-model we can assume balanced computation trees which in turn leads to easy plt-reductions. The advantage over the b-model is the established tight connection between the polynomial-time hierarchy and the Straubing-Thérien hierarchy, a well-studied hierarchy of regular languages. Glaßer [9] shows that such a connection does not hold for the b-model. This connection within the e-model makes it possible to exactly characterize leaf-language classes in the environment of NP.

In order to describe our results we have to define the levels of the Straubing-Thérien hierarchy (STH). In the scope of this paper it suffices to summarize that the STH is a hierarchy of levels that contain regular languages. We use a notation that already suggests a connection to the polynomial-time hierarchy (PH). A language belongs to level $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$ if it can be defined by a sentence of the logic FO[<] on words such that the sentence starts with an existential quantifier and has at most $k-1$ quantifier alternations. $\Pi_{\mathrm{k}}^{\mathrm{FO}}$ denotes the level of the complements of elements in $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$. $\Delta_{\mathrm{k}}^{\mathrm{FO}}$ denotes the intersection of $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$ and $\Pi_{\mathrm{k}}^{\mathrm{FO}}$. The formal definition can be found in the preliminaries.

Results: We start with observations that let us easily transfer the known BCSV-theorem to the new notion. Along these lines we show that the polynomial-time hierarchy (PH) is connected with the Straubing-Thérien hierarchy in the following sense: The e-class of level $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$ of the STH equals level Σ_k^{P} of the PH. Note that this leaves room for the possibility that languages outside $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$ form e-classes that are still contained in Σ_k^{P}. So even the e-class of a superset of $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$ might be equal to Σ_k^{P}. For the lower levels, however, we are able to rule out this possibility. This proves a substantially *tighter connection* between both hierarchies. For instance, under the reasonable assumption coUP $\not\subseteq$ NP, we show that the languages in Σ_1^{FO} are the only languages whose e-classes are contained in NP. Hence, under this assumption, a language belongs to Σ_1^{FO} if and only if its e-class is contained in NP. This connects Σ_1^{FO} and NP in the strongest possible way. We obtain several other strong relationships of this type, they are summarized in Table 1. In particular, we prove the first gap theorem for Δ_2^{P} (Corollary 6). This is possible by the e-model's tight connection to the STH, by the forbidden-pattern characterization of Σ_2^{FO} which was proved by Pin and Weil [14], and by the equality $\mathrm{Leaf}_{\mathrm{u}}^{\mathrm{p}}(\Delta_2^{\mathrm{FO}}) = \Delta_2^{\mathrm{P}}$ which was shown by Borchert, Schmitz, and Stephan [4] and Borchert et al. [3].

Some comments about the results in Table 1 are appropriate. First, they can be interpreted as gap theorems for leaf-language definability. For instance, the row about Σ_1^{FO} tells us that any e-class either is contained in NP or contains at least coUP. Hence, once an e-class becomes bigger than NP, its complexity jumps to at least NP$\cup$coUP. Second, there exist several evidences that classes in the columns 3–5 are not contained in the corresponding class of column 2. In any case there exist oracles relative to which this non-containment holds. Third, all classes in the first column are decidable, i.e., on input of a finite automaton A we can decide whether the language accepted by A belongs to the class. This allows a decidable and precise classification of e-classes under the assumption that the classes in the 4th column are not contained in the respective class in the 2nd column. On input of a regular language B (via its finite automaton) we can determine whether or not B's e-class is contained in the classes of the 2nd column.

With U we identify the class of all languages whose e-class is (robustly) contained in 1NP. A language belongs to U if and only if membership of a word can be expressed in terms of a unique occurrence of a substring and in terms of forbidden substrings. This shows that U is a class of regular languages. We prove a decidable characterization of U, a so-called forbidden-pattern characterization. It exactly reveals the structure in a finite automaton that is responsible for shifting a language outside U.

Table 1. Summary of the gap theorems, B is a language different from $\emptyset$ and Σ^*[1]

$\mathcal{C}$	$\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(\mathcal{C}) =$	if $B \notin \mathcal{C}$ then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$ contains	if $B \in \mathrm{REG} - \mathcal{C}$ then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$ contains	if $B \in \mathrm{SF} - \mathcal{C}$ then $\mathrm{Leaf}^{\mathrm{p}}_{\mathrm{e}}(B)$ contains
$\emptyset$	$\emptyset$	UP or coUP	NP, coNP, or $\mathrm{MOD}_p\mathrm{P}$ for a prime p	NP or coNP
Σ^{FO}_1	NP	coUP	coNP, co1NP, or $\mathrm{MOD}_p\mathrm{P}$ for a prime p	coNP or co1NP
Π^{FO}_1	coNP	UP	NP, 1NP, or $\mathrm{MOD}_p\mathrm{P}$ for a prime p	NP or 1NP
U	1NP	$\mathrm{UP} \vee \mathrm{UP}$ or $\mathrm{UP} \dot{\vee} \mathrm{coUP}$	$\mathrm{UP} \vee \mathrm{UP}$ or $\mathrm{UP} \dot{\vee} \mathrm{coUP}$	$\mathrm{UP} \vee \mathrm{UP}$ or $\mathrm{UP} \dot{\vee} \mathrm{coUP}$
coU	co1NP	$\mathrm{coUP} \wedge \mathrm{coUP}$ or $\mathrm{UP} \underset{\cdot}{\wedge} \mathrm{coUP}$	$\mathrm{coUP} \wedge \mathrm{coUP}$ or $\mathrm{UP} \underset{\cdot}{\wedge} \mathrm{coUP}$	$\mathrm{coUP} \wedge \mathrm{coUP}$ or $\mathrm{UP} \underset{\cdot}{\wedge} \mathrm{coUP}$
Δ^{FO}_2	Δ^{P}_2	–	$\mathrm{AU}\Sigma^{\mathrm{P}}_2$ or $\mathrm{AU}\Pi^{\mathrm{P}}_2$	$\mathrm{AU}\Sigma^{\mathrm{P}}_2$ or $\mathrm{AU}\Pi^{\mathrm{P}}_2$
Σ^{FO}_2	Σ^{P}_2	–	$\mathrm{AU}\Pi^{\mathrm{P}}_2$	$\mathrm{AU}\Pi^{\mathrm{P}}_2$
Π^{FO}_2	Π^{P}_2	–	$\mathrm{AU}\Sigma^{\mathrm{P}}_2$	$\mathrm{AU}\Sigma^{\mathrm{P}}_2$

Gap theorems for leaf-language definability are rather rare. With the following theorem we summarize the known results.

Theorem 1. *Let B be a nontrivial regular language.*

1. *[1] The u-class of B either is contained in* P*, or contains at least one of the following classes:* NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ *for some prime p.*
2. *[2] The u-class of B either is contained in* NP*, or contains at least one of the following classes:* coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ *for some prime p.*
3. *[15] The u-class of B either is contained in Σ^{P}_2, or contains $\mathrm{AU}\Pi^{\mathrm{P}}_2$.*
4. *[9] The b-class of B either is contained in* P*, or contains at least one of the following classes:* NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ *for some prime p.*
5. *[9] The b-class of B either is contained in* NP*, or contains at least one of the following classes:* coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ *for some prime p.*

2 Preliminaries

2.1 Basic Notions

We denote with NL, P, NP, coNP and PSPACE the standard complexity classes whose definitions can be found in any textbook on computational complexity. The class UP is the class of decision problems solvable by an NP machine such

[1] Some remarks about notations: $\mathcal{C} \vee \mathcal{D}$ (resp., $\mathcal{C} \dot{\vee} \mathcal{D}$) is the class of unions (resp., disjoint unions) of some $L_1 \in \mathcal{C}$ and some $L_2 \in \mathcal{D}$. From this, the operators $\wedge$ and $\underset{\cdot}{\wedge}$ are derived via DeMorgan's law. $\mathrm{AU}\Sigma^{\mathrm{P}}_2$ and $\mathrm{AU}\Pi^{\mathrm{P}}_2$ denote levels of the unambiguous polynomial-time hierarchy.

that if the input belongs to the language, exactly one computation path accepts and if the input does not belong to the language, all computation paths reject. Contrary, the class 1NP (also called US) is the class of decision problems solvable by an NP machine such that the input belongs to the language if and only if exactly one computation path accepts. For any $k > 1$, $\mathrm{MOD}_k\mathrm{P}$ is the class of decision problems solvable by an NP machine such that the number of accepting paths is divisible by k if and only if the input does not belong to the language. The characteristic function of a set A is χ_A. We assume that our alphabet Σ contains at least 2 letters. For a class of languages $\mathcal{C}$, co$\mathcal{C}$ is the class of complements of languages in $\mathcal{C}$.

Let $\preceq$ denote the usual subword relation, i.e., $v \preceq w$ if $v = v_1 \ldots v_n$ for letters $v_1, \ldots, v_n$ and $w \in \Sigma^* v_1 \Sigma^* v_2 \ldots \Sigma^* v_n \Sigma^*$. We write $v \prec w$ if $v \preceq w$ and $v \neq w$. For $k \geq 0$ we write $v \preceq_k w$ if v is a nonempty word that appears precisely k-times as a subword of w. In addition we define $\varepsilon \preceq_1 w$ for every word w. For $k \geq 0$ we write $v \preceq_{\geq k} w$ if there exists $l \geq k$ such that $v \preceq_l w$. For $k \geq 0$ and a finite set B of words $v_1, \ldots, v_{|B|}$ we write $B \preceq_k w$ if k can be written as $k = k_1 + \cdots + k_{|B|}$ such that $v_1 \preceq_{k_1} w,\ \ v_2 \preceq_{k_2} w,\ \ \ldots,\ \ v_{|B|} \preceq_{k_{|B|}} w$. So $v \preceq w$ if and only if there exists $k \geq 1$ such that $v \preceq_k w$. Also, $v \not\preceq w$ if and only if $v \preceq_0 w$. For example, it holds that $10 \preceq_3 1110$ and $\{0, 1, 10\} \preceq_7 1110$.

We call a language B nontrivial if $B \neq \emptyset$ and $B \neq \Sigma^*$.

Whenever we talk about a pair $(L, K) \subseteq \Sigma^*$ of languages, we assume that $L \subseteq \Sigma^*$ and $K \subseteq \Sigma^*$ are disjoint.

Definition 1. *Let $\mathcal{K}, \mathcal{M}$ be complexity classes. We define*

$$\mathcal{K} \vee \mathcal{M} =_{\mathrm{def}} \{A \cup B \,|\, A \in \mathcal{K}, B \in \mathcal{M}\},\ \ \mathcal{K} \wedge \mathcal{M} =_{\mathrm{def}} \mathrm{co}(\mathrm{co}\mathcal{K} \vee \mathrm{co}\mathcal{M}),$$
$$\mathcal{K} \dot{\vee} \mathcal{M} =_{\mathrm{def}} \{A \cup B \,|\, A \in \mathcal{K}, B \in \mathcal{M}, A \cap B = \emptyset\},\ \ \mathcal{K} \underset{\cdot}{\wedge} \mathcal{M} =_{\mathrm{def}} \mathrm{co}(\mathrm{co}\mathcal{K} \dot{\vee} \mathrm{co}\mathcal{M}).$$

The Unambiguous Alternation Hierarchy. Niedermeier and Rossmanith [12] introduced the *unambiguous alternation hierarchy.* For any complexity class $\mathcal{C}$, define $\exists^{\mathrm{u}}{\cdot}\mathcal{C}$ as the class of languages L such that there exist a polynomial p and $L' \in \mathcal{C}$ such that for all x the following holds: If x is in L, there exists exactly one $y \in \Sigma^{=p(|x|)}$ such that $(x, y) \in L'$. If x is not in L, there exists no $y \in \Sigma^{=p(|x|)}$ such that $(x, y) \in L'$. Similarly, $\forall^{\mathrm{u}}{\cdot}\mathcal{C} =_{\mathrm{def}} \mathrm{co}\exists^{\mathrm{u}}{\cdot}\mathrm{co}\mathcal{C}$.

Definition 2 (attributed to unpublished work of Hemaspaandra [12]). $\mathrm{AU\Sigma^P_0} = \mathrm{AU\Pi^P_0} =_{\mathrm{def}} \mathrm{P}$, $\mathrm{AU\Sigma^P_{k+1}} =_{\mathrm{def}} \exists^{\mathrm{u}}{\cdot}\mathrm{AU\Pi^P_k}$ *for* $k \geq 0$, $\mathrm{AU\Pi^P_{k+1}} =_{\mathrm{def}} \forall^{\mathrm{u}}{\cdot}\mathrm{AU\Sigma^P_k}$ *for* $k \geq 0$.

Spakowski and Tripathi [16] construct an oracle relative to which for every $n \geq 1$, level n of the unambiguous alternation hierarchy is not contained in Π^{P}_n.

The Straubing-Thérien hierarchy. Starfree languages are regular languages that can be build from single letters by using Boolean operations and concatenation. Let SF denote the class of starfree languages. Brzozowski and Cohen [8,6] introduced the *dot-depth hierarchy* which measures the complexity of starfree languages in terms of necessary alternations between Boolean operations and concatenation. Straubing and Thérien [17,19,18] introduced a modification that is more appropriate for the algebraic theory of languages, but still covers the important aspects of the dot-depth hierarchy. This hierarchy is called Straubing-Thérien hierarchy (STH).

Perrin and Pin [13] proved a logical characterization of the STH. We use this characterization as definition, since it uses an easy logic on words and it shows nice parallels to the definition of the polynomial-time hierarchy. Formulas of the first-order logic FO[<] consist of first-order quantifiers, Boolean operators, the binary relation symbol $<$, and unary relation symbols π_a for each letter a. A sentence ϕ is satisfied by a word w if ϕ evaluates to true where variables are interpreted as positions in w and $\pi_a x$ is interpreted as "letter a appears at position x in w". A language B is FO[<] definable if there exists a sentence ϕ such that for all words w, $w \in L$ if and only if ϕ is satisfied by w. A $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$-sentence (resp., $\Pi_{\mathrm{k}}^{\mathrm{FO}}$-sentence) is a sentence of FO[<] that is in prenex normal form, that starts with an existential (resp., universal) quantifier, and that has at most $k-1$ quantifier alternations. A language belongs to the class $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$ (resp., $\Pi_{\mathrm{k}}^{\mathrm{FO}}$) of the STH if it can be defined by a $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$-sentence (resp., $\Pi_{\mathrm{k}}^{\mathrm{FO}}$-sentence). $\Delta_{\mathrm{k}}^{\mathrm{FO}}$ denotes the intersection of $\Sigma_{\mathrm{k}}^{\mathrm{FO}}$ and $\Pi_{\mathrm{k}}^{\mathrm{FO}}$.

3 Machines with Computation Trees Having ε-Leaves

We introduce the e-model of leaf languages which is inspired by the observation that rejecting paths of nondeterministic computations act as neutral elements. We allow nondeterministic transducers not only to output single letters, but also to output the empty word ε. After the formal definition we introduce pte-reducibility which allows us to formulate and prove an analogue of the BCSV-theorem. Furthermore, we show that the e-model connects the polynomial-time hierarchy with the Straubing-Thérien hierarchy.

For a finite alphabet Σ and $a \notin \Sigma$, we define a homomorphism $h_{\Sigma,a} : (\Sigma \cup \{a\})^* \to \Sigma^*$ by $h_{\Sigma,a}(b) =_{\mathrm{def}} b$ for $b \in \Sigma$ and $h_{\Sigma,a}(a) =_{\mathrm{def}} \varepsilon$.

Definition 3. *Let $(L, K) \subseteq \Sigma^*$. The class* $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L, K)$ *consists of all languages A for which there exists a nondeterministic polynomial time transducer M producing on every computation path a symbol from Σ or the empty word ε such that the following holds:*

$$x \in A \Rightarrow \beta_M(x) \in L, x \notin A \Rightarrow \beta_M(x) \in K.$$

For $(L, K) \subseteq \Sigma^*$, if $K = \Sigma^* - L$, we will often use $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L)$ as abbreviation for $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L, K)$. In these cases, we will make clear what alphabet we use for L. Notice the it makes no difference whether we use balanced or unbalanced computation trees. So for convenience we may assume that paths not only can output single letters, but arbitrary words.

Example 1. 1. $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(11^*, \varepsilon) = \mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(0^*1(0 \vee 1)^*, 0^*) = \mathrm{NP}$.
2. Let $L =_{\mathrm{def}} \{1\} \subseteq \{0, 1\}^*$. Then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L) = \mathrm{1NP}$.
3. $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(1, \varepsilon) = \mathrm{UP}$.

A function g is computable in polylogarithmic time if there exists $k \geq 1$ such that $g(x)$ can be computed in time $\mathcal{O}(\log^k |x|)$ by a Turing-machine which accesses the input as an oracle.

Definition 4. *Let $(L, K) \subseteq \Sigma_1^*, (L', K') \subseteq \Sigma_2^*$ and $a \notin \Sigma_1^* \cup \Sigma_2^*$. Then $(L, K) \leq_{\mathrm{m}}^{\mathrm{pte}} (L', K')$ if and only if there exists a function $f : (\Sigma_1 \cup \{a\})^* \to (\Sigma_2 \cup \{a\})^*$ such that*

- *there exist functions $g : \big((\Sigma_1 \cup \{a\})^* \times \mathbb{N}\big) \to \Sigma_2 \cup \{a\},\ h : (\Sigma_1 \cup \{a\})^* \to \mathbb{N}$ computable in polylogarithmic time such that for all $x \in (\Sigma_1 \cup \{a\})^*$, $f(x) = g(x, 1)g(x, 2) \ldots g(x, h(x))$,*
- *for all $x \in (\Sigma_1 \cup \{a\})^*$, $\big(h_{\Sigma_1,a}(x) \in L \Rightarrow h_{\Sigma_2,a}(f(x)) \in L'\big)$,*
- *for all $x \in (\Sigma_1 \cup \{a\})^*$, $\big(h_{\Sigma_1,a}(x) \in K \Rightarrow h_{\Sigma_2,a}(f(x)) \in K'\big)$.*

If $(L, K) \leq_{\mathrm{m}}^{\mathrm{pte}} (L', K')$ and $K = \Sigma_1^* - L$ and $K' = \Sigma_2^* - L'$, we write $L \leq_{\mathrm{m}}^{\mathrm{pte}} L'$ as abbreviation.

We obtain the following BCSV-theorem for the e-model.

Theorem 2. *Let $(L, K) \subseteq \Sigma_1^*$ and $(L', K') \subseteq \Sigma_2^*$. Then the following statements are equivalent:*

1. *$(L, K) \leq_{\mathrm{m}}^{\mathrm{pte}} (L', K')$.*
2. *For all oracles O it holds that $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L, K)^O \subseteq \mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L', K')^O$.*

The next theorem shows a connection between the STH and the PH via the e-model. A similar connection for the existing b- and u-models was proved by Hertrampf et al. [11], Burtschick and Vollmer [7], and Borchert et al. [3].

Theorem 3. *For $k \geq 1$, it holds that $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(\Sigma_{\mathrm{k}}^{\mathrm{FO}}) = \Sigma_k^{\mathrm{P}}$, $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(\Pi_{\mathrm{k}}^{\mathrm{FO}}) = \Pi_k^{\mathrm{P}}$, $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(\Delta_{\mathrm{k}}^{\mathrm{FO}}) = \Delta_k^{\mathrm{P}}$.*

4 Gap Theorems for NP, Δ_2^{P}, and Σ_2^{P}

In this section we use existing forbidden-pattern characterizations to obtain lower bounds for certain e-classes. From this we derive gap theorems for NP, Δ_2^{P}, and Σ_2^{P}. A summary of these results can be found in Table 1. Pin and Weil [14] proved the following forbidden-pattern characterization of level Σ_1^{FO} of the STH.

Proposition 1 ([14]). *The following holds for any language A:*

$$\begin{aligned} A \in \Sigma_1^{\mathrm{FO}} &\Leftrightarrow \forall v, w \in \Sigma^* \big(v \preceq w \Rightarrow \chi_A(v) \leq \chi_A(w)\big) \\ &\Leftrightarrow \forall v, w \in \Sigma^* \Big(\forall a \in \Sigma \big(\chi_A(vw) \leq \chi_A(vaw)\big)\Big) \end{aligned}$$

This characterization enables us to prove lower bounds for the e-class of languages outside Σ_1^{FO}. In combination with Theorem 3 we obtain a gap theorem for NP (Corollary 1).

Theorem 4. *Let A be an arbitrary language.*

1. *If $A \notin \Sigma_1^{\mathrm{FO}}$, then $\mathrm{coUP} \subseteq \mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$.*
2. *If $A \in \mathrm{REG} - \Sigma_1^{\mathrm{FO}}$, then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$ contains at least one of the following classes:* coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ *for a prime p.*
3. *If $A \in \mathrm{SF} - \Sigma_1^{\mathrm{FO}}$, then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$ contains at least one of the following classes:* coNP, co1NP.

Corollary 1. *Let B be a nontrivial language.*

1. *The e-class of B either is contained in* NP*, or contains* coUP.
2. *If $B \in \mathrm{REG}$, then the e-class of B either is contained in* NP*, or contains at least one of the following classes:* coNP, co1NP, $\mathrm{MOD}_p\mathrm{P}$ *for a prime p.*
3. *If $B \in \mathrm{SF}$, then the e-class of B either is contained in* NP*, or contains at least one of the following classes:* coNP, co1NP.

Now we can prove general lower bounds for e-classes. In particular, no complexity class below UP is definable with this concept.

Corollary 2. *Let A be a nontrivial language.*

1. *$\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$ contains at least one of the following classes:* UP, coUP.
2. *If $A \in \mathrm{REG}$, then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$ contains at least one of the following classes:* NP, coNP, $\mathrm{MOD}_p\mathrm{P}$ *for a prime p.*
3. *If $A \in \mathrm{SF}$, then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$ contains at least one of the following classes:* NP, coNP.

Under reasonable assumptions there is no regular A such that A's e-class lies strictly between coNP and 1NP. By symmetry, the same holds for NP and co1NP.

Corollary 3. *Let $A \in \mathrm{REG}$ be a nontrivial language. Assume $\mathrm{NP} \not\subseteq \mathrm{1NP}$ and $\mathrm{MOD}_p\mathrm{P} \not\subseteq \mathrm{1NP}$ for all primes p. Then the following implication holds.*

$$\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A) \subsetneq \mathrm{1NP} \Rightarrow \mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A) \subseteq \mathrm{coNP}.$$

Starting with a forbidden-pattern characterization for Σ_2^{FO} [14] we develop a lower bound for the e-class of Σ_2^{FO}. Again, this yields a gap theorem, this time for Σ_2^{P} (Corollary 4).

Theorem 5. *If $A \in \mathrm{REG} - \Sigma_2^{\mathrm{FO}}$, then $\mathrm{AU}\Pi_2^{\mathrm{P}} \subseteq \mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$.*

Corollary 4. *Let B be a nontrivial, regular language. The e-class of B either is contained in Σ_2^{P}, or contains $\mathrm{AU}\Pi_2^{\mathrm{P}}$.*

In addition, Theorem 5 gives us a lower bound for the e-class of Δ_2^{FO}:

Corollary 5. *If $A \in \mathrm{REG} - (\Sigma_2^{\mathrm{FO}} \cap \Pi_2^{\mathrm{FO}})$, then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(A)$ contains at least one of the following classes: $\mathrm{AU}\Pi_2^{\mathrm{P}}$, $\mathrm{AU}\Sigma_2^{\mathrm{P}}$.*

The following is the first gap theorem for Δ_2^{P}. It holds for both the u- and the e-model.

Corollary 6. *Let B be a nontrivial, regular language.*

1. *The e-class of B either is contained in Δ_2^{P}, or contains at least one of the following classes:* $\mathrm{AU}\Sigma_2^{\mathrm{P}}$, $\mathrm{AU}\Pi_2^{\mathrm{P}}$.
2. *The u-class of B either is contained in Δ_2^{P}, or contains at least one of the following classes:* $\mathrm{AU}\Sigma_2^{\mathrm{P}}$, $\mathrm{AU}\Pi_2^{\mathrm{P}}$.

5 A Gap Theorem for 1NP

In view of the gap theorems for NP and coNP (Corollary 1) it becomes evident that the classes 1NP and MOD$_p$P play an important role since they appear as lower bounds. In this section we analyze 1NP in detail and prove a gap theorem for this class. This case is more challenging since we cannot utilize an existing forbidden-pattern characterization. With Theorem 6 we give such a characterization for the class of languages corresponding to 1NP. Additionally, this theorem shows that with this class we have in fact identified *all* languages whose e-class is robustly contained in 1NP. This lets us derive a gap theorem for 1NP. For a given language L, we define the following conditions:

P1: There exist words $u \in L$, $v \notin L$, and $w \in L$ such that $u \preceq v \preceq w$.

P2: There exist $k \geq 2$ and nonempty words $u, v, w \in L$ such that $\{u, v\} \preceq_k w$ and $\forall x\big(((x \prec u \vee x \prec v) \Rightarrow x \notin L\big)$.

We interpret the patterns P1 and P2 as forbidden patterns and define a class of languages U of languages which neither fulfill P1 nor P2:

$$\mathrm{U} =_{\mathrm{def}} \{L \,|\, \text{P1 and P2 fail for } L\}$$

We will later on see that U is in fact a class of regular languages, and precisely characterizes the class 1NP in the e-model of leaf-languages. The next lemma shows that the e-class of a language which fulfills P1 or P2 is quite powerful.

Lemma 1. *Let $L \subseteq \Sigma^*$ be a language.*
1. If L satisfies P1, then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L) \supseteq \mathrm{UP} \dot{\vee} \mathrm{coUP}$.
2. If L satisfies P2, then $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L) \supseteq \mathrm{UP} \vee \mathrm{UP}$.

The next lemma gives simple languages that define the classes 1NP, $\mathrm{UP} \dot{\vee} \mathrm{coUP}$, and $\mathrm{UP} \vee \mathrm{UP}$ in terms of leaf-languages.

Lemma 2. *1.* $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(1, (\varepsilon \vee 111^*)) = 1\mathrm{NP}$, *2.* $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}((\varepsilon \vee 12), 2) = \mathrm{UP} \dot{\vee} \mathrm{coUP}$, *3.* $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}((1 \vee 2 \vee 12), \varepsilon) = \mathrm{UP} \vee \mathrm{UP}$.

In order to show that for any language L, fulfillment of P1 suffices for the class $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L)$ to be not robustly contained in 1NP, we first prove that languages characterizing $\mathrm{UP} \dot{\vee} \mathrm{coUP}$ cannot be pte-reduced to languages characterizing 1NP.

Lemma 3. $((\varepsilon \vee 12), 2) \not\leq_{\mathrm{m}}^{\mathrm{pte}} (1, (\varepsilon \vee 111^*))$.

Theorem 2 enables us to translate this statement into an oracle separation:

Lemma 4. *There exists an oracle O such that* $\mathrm{UP} \dot{\vee} \mathrm{coUP} \not\subseteq 1\mathrm{NP}^O$.

We now prove that languages characterizing UP $\vee$ UP cannot be pte-reduced to languages characterizing 1NP. This is a step towards showing that for any language L, fulfillment of P2 suffices for the class $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L)$ not to be robustly contained in 1NP. Again, we achieve this by applying Theorem 2 to Lemma 5.

Lemma 5. $((1 \vee 2 \vee 12), \varepsilon) \not\leq_{\mathrm{m}}^{\mathrm{pte}} (1, (\varepsilon \vee 111^*))$.

Lemma 6. *There exists an oracle O such that* $\mathrm{UP} \vee \mathrm{UP} \not\subseteq 1\mathrm{NP}^O$.

So e-classes of languages outside U are not in 1NP. The next theorem gives the characterization of U, the class that precisely corresponds to 1NP in the e-model.

Theorem 6. *The following statements are equivalent for any language $L \subseteq \Sigma^*$.*

1. $L \in \mathcal{R}^{pte}(1)$*, the pte-closure of* $\{1\}$.
2. *For all oracles O it holds that* $\mathrm{Leaf}_{\mathrm{e}}^{\mathrm{p}}(L)^O \subseteq 1\mathrm{NP}^O$.
3. $L \in \mathrm{U}$*, that means both conditions, P1 and P2, fail for L.*
4. *There exist finite sets $A, B \subseteq \Sigma^*$ such that*
   $L = \{w \mid A \preceq_1 w \text{ and } (\forall v \in B)[v \not\preceq w]\}$.

Observe that due to the characterization of U given by Theorem 6.4, we immediately obtain that U only contains regular languages. We can now formulate the new gap theorem.

Theorem 7. *Let L be a nontrivial language. If $L \in \mathrm{U}$, then the e-class of L is contained in* 1NP*. If $L \notin \mathrm{U}$, then the e-class of L contains* $\mathrm{UP} \dot{\vee} \mathrm{coUP}$ *or* $\mathrm{UP} \vee \mathrm{UP}$.

Acknowledgments

We thank Bernd Borchert, Victor Selivanov, and Klaus W. Wagner for very interesting discussions and many helpful suggestions.

References

1. B. Borchert. On the acceptance power of regular languages. *Theoretical Computer Science*, 148:207–225, 1995.
2. B. Borchert, D. Kuske, and F. Stephan. On existentially first-order definable languages and their relation to NP. *Theoretical Informatics and Applications*, 33:259–269, 1999.
3. B. Borchert, K. Lange, F. Stephan, P. Tesson, and D. Thérien. The dot-depth and the polynomial hierarchies correspond on the delta levels. *International Journal of Foundations of Computer Science*, 16(4):625–644, 2005.

4. B. Borchert, H. Schmitz, and F. Stephan. Unpublished manuscript, 1999.
5. D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.
6. J. A. Brzozowski. Hierarchies of aperiodic languages. *RAIRO Inform. Theor.*, 10:33–49, 1976.
7. H.-J. Burtschick and H. Vollmer. Lindström quantifiers and leaf language definability. *International Journal of Foundations of Computer Science*, 9:277–294, 1998.
8. R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5:1–16, 1971.
9. C. Glaßer. Polylog-time reductions decrease dot-depth. In *Proceedings 22nd Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*. Springer Verlag, 2005.
10. C. Glaßer, M. Ogihara, A. Pavan, A. L. Selman, and L. Zhang. Autoreducibility, mitoticity, and immunity. In *Proceedings 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 387–398. Springer-Verlag, 2005.
11. U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.
12. R. Niedermeier and P. Rossmanith. Unambiguous computations and locally definable acceptance types. *Theoretical Computer Science*, 194(1-2):137–161, 1998.
13. D. Perrin and J. E. Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32:393–406, 1986.
14. J. E. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory of computing systems*, 30:383–422, 1997.
15. H. Schmitz. *The Forbidden-Pattern Approach to Concatenation Hierarchies*. PhD thesis, Fakultät für Mathematik und Informatik, Universität Würzburg, 2001.
16. H. Spakowski and R. Tripathi. On the power of unambiguity in alternating machines. In *Proceedings 15th International Conference on Fundamentals of Computation Theory*, *Lecture Notes in Computer Science* 3623, pages 125–136, 2005.
17. H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150, 1981.
18. H. Straubing. Finite semigroups varieties of the form V * D. *J. Pure Appl. Algebra*, 36:53–94, 1985.
19. D. Thérien. Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14:195–208, 1981.
20. N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.
21. K. W. Wagner. Leaf language classes. In *Proceedings International Conference on Machines, Computations, and Universality*, volume 3354 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

Completeness of Global Evaluation Logic

Sergey Goncharov[1], Lutz Schröder[1,2], and Till Mossakowski[1,2]

[1] Department of Computer Science, University of Bremen, Germany
[2] DFKI Lab Bremen, Germany

Abstract. Monads serve the abstract encapsulation of side effects in semantics and functional programming. Various monad-based specification languages have been introduced in order to express requirements on generic side-effecting programs. A basic role is played here by global evaluation logic, concerned with formulae which may be thought of as being universally quantified over the state space; this formalism is the fundament of more advanced logics such as monad-based Hoare logic or dynamic logic. We prove completeness of global evaluation logic for models in cartesian categories with a distinguished Heyting algebra object.

Introduction

Monads form the basis of the abstract treatment of side effects in modern functional programming languages and in associated specification languages. The computational significance of monads was first noticed by Moggi [7]. They were subsequently incorporated in the design of the functional programming language Haskell [11] as the principal means of dealing with impure features such as Input/Output, but also as a convenient mode of expression for state threading, e.g. in parsing [3]. Monads are by now also well-established as a tool in semantics (cf. [15] for a recent example, and [4] for an application to the semantics of Java). The advantage of the abstraction of notions of side effect as monads is twofold: firstly, one obtains a generic framework that can be instantiated with many particular monads, thus promoting the reuse of concepts and indeed programs. Secondly, side effects are cleanly encapsulated and equipped with a clearly delineated scope [9, 2].

The use of monads in programming has led to the design of various specification logics for monadic programs, including Pitts' evaluation logic [12], Moggi's globalized version of evaluation logic [8], a monad-based Hoare logic [13], and a monad-based dynamic logic related to evaluation logic [14]. The basic tool in the semantics of the latter two logics is *global evaluation logic*, concerned with statements of the nature 'after execution of a given program from *any* initial state, it is the case that . . . ' (referred to as *global dynamic judgements* in [14]). This logic is closely related to Moggi's global semantics of evaluation logic (which is in fact markedly dissimilar from the originally intended *local* semantics of evaluation logic [12]).

Unlike some of the more complex logics, global evaluation logic can be interpreted over arbitrary monads, under only quite rudimentary requirements on the base category. In this work, we make this semantics explicit, thus detaching the calculus from the more complex framework of the specification language HasCasl (i.e. higher order logic of partial functions) in which it was originally presented [13, 14]. We then prove completeness of global evaluation logic for the said class of models, viz. over cartesian categories with a distinguished Heyting algebra object.

This work forms part of an ongoing effort to establish completeness results for monad-based computational logics. A further result in this direction is the completeness of monad-based dynamic logic [10]. While no completeness result for the local semantics of evaluation logic is given in [12], Moggi does prove completeness for his global version of evaluation logic [8]. The crucial difference between the latter result and the one presented here is that the logic considered in [8] includes the entire equational theory of monads, i.e. the meta-language introduced in [7]. This fact plays a central role in Moggi's completeness proof, which relies on constructing a term model using the equations of the meta-language. In contrast to this, our global evaluation logic talks purely about the observable behaviour of programs, and excludes statements about actual equality of programs (we do admit an underlying equational theory, which however shows up only in rules expressing Leibniz equalities). It may be argued that this amounts to a better encapsulation of program behaviour, and is closer in spirit to program logics such as Hoare logic, which also talk only about properties of the state space affected by program execution rather than about equality of programs. In consequence, the term model construction in our completeness proof is more involved: we cannot rely on treating equality by means of an equational calculus, but need to introduce a Leibniz equality on terms. An important tool in the model construction is a normalization result for program sequences, which may be of independent interest.

The material is organized as follows. We recall the basics of monad-based programming in Sect. 1. In Sect. 2, we define the syntax and semantics of global evaluation logic, indicate how global evaluation logic is used in more advanced program logics, and discuss the relation of our work to [8] in more detail. The proof calculus of the logic is presented in Sect. 3. Section 4 is devoted to the completeness proof of this calculus.

1 Monads for Computations

Intuitively, a monad associates to each type A a type TA of computations of type A; a function with side effects that takes inputs of type A and returns values of type B is, then, just a function of type $A \to TB$. This approach abstracts away from particular notions of computation such as store, non-determinism, non-termination etc.; a surprisingly large amount of reasoning can be carried out without commitment to a particular type of side effect.

A monad on a category $\mathbf{C}$ can be defined as a *Kleisli triple* $\mathbb{T} = (T, \eta, \_^*)$, where $T : \mathrm{Ob}\,\mathbf{C} \to \mathrm{Ob}\,\mathbf{C}$ is a function, the *unit* η is a family of morphisms $\eta_A : A \to TA$, and $\_^*$ assigns to each morphism $f : A \to TB$ a morphism $f^* : TA \to TB$ such that

$$\eta_A^* = id_{TA}, \quad f^*\eta_A = f, \quad \text{and} \quad g^*f^* = (g^*f)^*.$$

This description is equivalent to the more familiar one via an endofunctor with unit and multiplication [6].

In order to support a language with finitary operations and multi-variable contexts (see below), one needs a further ingredient: a monad is called *strong* if it is equipped with a natural transformation

$$t_{A,B} : A \times TB \to T(A \times B)$$

called *strength*, subject to certain coherence conditions [7].

Example 1. [7] Computationally relevant monads on **Set** include

- Stateful computations with possible non-termination: $TA = (S \to ? \ (A \times S))$, where S is a fixed set of states and $\_ \to ? \ \_$ denotes the partial function type.
- Non-determinism: $TA = \mathcal{P}(A)$, where $\mathcal{P}$ denotes the covariant power set functor (which takes a map $f : A \to B$ to the map $\mathcal{P}(f) : \mathcal{P}(A) \to \mathcal{P}(B)$, $C \mapsto f[C]$).
- Exceptions: $TA = A + E$, where E is a fixed set of exceptions.
- Interactive input: TA is the smallest fixed point $\mu\gamma.\, A + (U \to \gamma)$, where U is a set of input values.
- Non-deterministic stateful computations: $TA = \mathcal{P}(S \to (A \times S))$, where, again, S is a fixed set of states.
- Continuations: $TA = (A \to R) \to R$, where R is a type of *results*.

Reasoning about a category equipped with a strong monad is greatly facilitated by the fact that proofs can be conducted in a *computational meta-language* with the following features, introduced in [7]:

- A type constructor T; TA is the type of A-valued *programs* or *computations*;
- a polymorphic operator ret : $A \to TA$ corresponding to the unit;
- a binding construct, which we denote in Haskell's do style: terms of the form

  $$\text{do } x \leftarrow p;\ q$$

  are interpreted by means of the tensorial strength and Kleisli composition (cf. [7] for details). Intuitively, do $x \leftarrow p;\ q$ first computes p, whose result is assigned to the bound variable x, and then computes q (which may depend on x). Binding is *associative*, i.e. one has

  $$\text{do } y \leftarrow (\text{do } x \leftarrow p;\ q);\ r = \text{do } x \leftarrow p;\ \text{do } y \leftarrow q;\ r$$

  if r does not contain x. We denote nested do expressions do $x \leftarrow p;$ do $y \leftarrow q;\ \dots$ by do $x \leftarrow p; y \leftarrow q;\ \dots$. Repeated nestings do $x_1 \leftarrow p_1, \dots, x_n \leftarrow p_n;\ q$ are denoted in the form do $\bar{x} \leftarrow \bar{p};\ q$. Term fragments of the form $\bar{x} \leftarrow \bar{p}$ are called *program sequences.* Variables x_i that do not appear later on may be omitted from the notation. Terms that differ only in the names of bound variables are equal (α-equivalence). The scope of a binding extends as far as possible.

– besides the associative law mentioned above, *unit laws* stating that

$$\begin{aligned}&(\text{do } x \leftarrow y;\ \text{ret}\, x) = y \qquad \text{and}\\ &(\text{do } x \leftarrow \text{ret}\, a;\ p) = p[a/x].\end{aligned}$$

Terms are generally formed in a context of variables with assigned types; however, we will omit contexts from the notation.

A program p is called *stateless* if it factors through ret, i.e. if it is just a value inserted into the monad. In [14], we also introduce a notion of *deterministically side-effect free (dsef)* monadic value. All stateless programs are dsef, but not vice versa. For example, in the state monad, a program that reads the state but does not modify it is dsef but not stateless.

2 Global Evaluation Logic

Global evaluation logic is a simple specification language for monadic programs. It is related to (monad-based) dynamic logic in that it features a modality that encapsulates the effect of executing a program; however, unlike in dynamic logic this modality is *global*, i.e. implicitly quantified over all states. Thus, the formulae of global evaluation logic are of the form

$$[\![x \leftarrow p]\!]\, \phi,$$

where $p : TA$ is a monadic program and ϕ is a stateless formula possibly containing the free variable x, to be read informally as *for all states s, execution of p in state s yields a result x satisfying ϕ.*

Global evaluation logic is used chiefly as a means of encoding or defining more advanced logics. In the monad-based Hoare calculus [13], Hoare triples are defined by

$$\{\varphi\}\ x \leftarrow p\ \{\psi\} :\equiv [\![a \leftarrow \varphi; x \leftarrow p; b \leftarrow \psi]\!]\, (a \rightarrow b).$$

Here ψ may contain x, and φ and ψ are *dynamic formulae*, i.e. computations of truth values which are dsef, in particular may access but not modify the state. The Hoare triple $\{\varphi\}\ x \leftarrow p\ \{\psi\}$ has, in typical examples, the expected meaning *if program p is executed in a state satisfying φ, terminating in some state s, then s and the result x of p satisfy ψ.* Further examples of encodings of program logics in a logic related to global evaluation logic are given in [8]. Moreover, monad-based dynamic logic [14], which like Pitts' local evaluation logic [12] is concerned with nestable modal operators equipped with a *state-dependent* semantics, is axiomatised in terms of global evaluation logic. Completeness of monad-based dynamic logic is shown in [10] without recourse to more basic completeness results. The pending completeness proof for the monad-based Hoare calculus, however, is expected to make use of completeness of global evaluation logic.

To begin, we separate the definition of the syntax and semantics of global evaluation logic from the HasCasl framework originally used in its formulation [14]. The presentation of the logic is slightly more general than in [14] in that we admit an underlying equational theory.

Given a set S of basic types, the set of *types* A is generated by the grammar

$$A ::= 1 \mid \Omega \mid TA \mid A \times A \mid S.$$

A type of the form TA is called a *computational type*. A type is called *T-free* if it does not contain occurrences of T. A signature $\Sigma = (S, F)$ consists of a set S of basic types and a set F of operation symbols $f : A \to B$, where A and B are types over S, *with A T-free*. The latter condition means that we exclude user-defined control structures; the design of a complete logic that includes such control structures is the subject of future research. The type Ω serves as a type of truth values; we assume that Σ contains operations on Ω representing the logical connectives.

A signature gives rise to a notion of *term* (in a context $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ of type assignments for variables, which we mostly omit from the notation) in the usual way, with term formation rules including variable introduction, application of basic operators, formation of pairs $\langle s, t\rangle : A \times B$, projection terms $\mathit{fst}(t)$, $\mathit{snd}(t)$, the distinguished term $*$ inhabiting the unit type 1, and the return and binding operations of the computational metalanguage. We denote the set of free variables occurring in a term t (or, later, a formula) by $FV(t)$. An *equation* is a pair of terms of the same type (in a common context).

An *equational theory* $\mathcal{E}$ is a pair (Σ, E), where Σ is a signature and E is a set of equations over Σ. We assume that $\mathcal{E}$ contains the Heyting algebra axioms for Ω (if classical reasoning is required, one may extend these to the axioms for a Boolean algebra). We write $\mathcal{E} \vdash s = t$ if $s = t$ can be derived from $\mathcal{E}$ using standard equational reasoning (this includes equations for product types involving terms of the computational metalanguage such as $\mathit{snd}\langle(\mathrm{do}\ x \leftarrow p;\ q), \mathrm{ret}\, a\rangle = \mathrm{ret}\, a$, but *excludes* the equations of the computational metalanguage, i.e. associativity and the two unit laws).

A *model* of a signature consists of a cartesian category $\mathbf{C}$ (i.e. a category with finite products) equipped with a distinguished object Ω and a strong monad T, together with interpretations of the basic types as objects in $\mathbf{C}$ and interpretations of the basic operations as morphisms in $\mathbf{C}$, referred to slightly inaccurately just as T. This extends to an interpretation $[\![A]\!]_T \in \mathrm{Ob}\,(\mathbf{C})$ for every type A by interpreting product types and unit types by the cartesian structure of $\mathbf{C}$, Ω as the distinguished object Ω, and computational types TA by applying the monad T. Then, a context $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$ is interpreted as the product $[\![\Gamma]\!]_T = \prod_{i=1}^{n} [\![A_i]\!]_T$. The interpretation of a basic operation $f : A \to B$ is required to be a morphism $[\![A]\!]_T \to [\![B]\!]_T$.

We then obtain, for every term $t : A$ in context Γ, an interpretation as a morphism $[\![t]\!]_T : [\![\Gamma]\!]_T \to [\![A]\!]_T$, where variables are interpreted as projections, pairing, projections, and $*$ are interpreted using the cartesian structure, application of basic operations is composition, and return and binding are interpreted as prescribed by T. A *model* of an equational theory $\mathcal{E} = (\Sigma, E)$ is a model of Σ that *satisfies* every equation $s = t$ in E in the sense that $[\![s]\!]_T = [\![t]\!]_T$. The equations in $\mathcal{E}$ imply that in every model of $\mathcal{E}$, the object Ω is a *Heyting algebra object*, i.e. its hom-functor $\mathit{hom}(\_, \Omega)$ factors through Heyting algebras.

As indicated above, a *formula* of *global evaluation logic* is an expression of the form $[\bar{x} \leftarrow \bar{p}]\, \phi$, where $p_i : TA_i$ and $\phi : \Omega$ are terms. We say that a model T *satisfies* $[\bar{x} \leftarrow \bar{p}]\, \phi$, and write $T \models [\bar{x} \leftarrow \bar{p}]\, \phi$, if

$$[\![\text{do } \bar{x} \leftarrow \bar{p};\ \text{ret}\langle \bar{x}, \phi\rangle]\!]_T = [\![\text{do } \bar{x} \leftarrow \bar{p};\ \text{ret}\langle \bar{x}, \top\rangle]\!]_T.$$

Moreover, T satisfies a set Φ of formulae, written $T \models \Phi$, if $T \models P$ for every $P \in \Phi$.

Definition 2. We say that the monad (or model) T is *simple* if satisfaction of $[\bar{x} \leftarrow \bar{p}]\, \phi$ is equivalent to

$$[\![\text{do } \bar{x} \leftarrow \bar{p};\ \text{ret}\, \phi]\!]_T = [\![\text{do } \bar{x} \leftarrow \bar{p};\ \text{ret}\, \top]\!]_T.$$

The calculus of Section 3, for which we prove completeness, is sound only for simple models.

Example 3. In the monads of Example 1, satisfaction of $[x \leftarrow p]\, \phi$, where $p : TA$, amounts to the following (we freely omit semantic brackets from the notation):

- *stateful computation*: terminating execution of p from any initial state yields a result x satisfying ϕ;
- *non-determinism*: all values x in $p \in \mathcal{P}(A)$ satisfy ϕ;
- *exceptions*: if p terminates normally, then its result x satisfies ϕ;
- *interactive input*: the value x eventually produced by p after some combination of inputs always satisfies ϕ;
- *non-deterministic stateful computation*: all possible results x obtained by execution of p from any initial state satisfy ϕ;
- *continuations*: for $k : A \to R$, $p\ k$ depends only on the restriction of k to the set of values $x : A$ satisfying ϕ.

All these monads except the continuation monad are simple [14].

Remark 4. A particularly troublesome case w.r.t. monad-based specification logics is the continuation monad (Example 1), the main problem being that the continuation monad has too few deterministically side-effect free computations — e.g. over **Set**, all its dsef computations are already stateless. Consequently, the continuation monad does not admit dynamic logic [14], and Hoare logic over the continuation monad, while sound, does not express any interesting properties, as pre- and postconditions are required to be dsef. Global evaluation logic, however, has a rather sensible interpretation over the continuation monad (Example 3). Thus, it appears feasible to take global evaluation logic as the starting point for the design of a program logic for the continuation monad. As the continuation monad fails to be simple, this means that there is some interest in the pending issue of finding a complete proof system for our global evaluation logic without the simplicity assumption. Note that using the equation in Def. 2 as an alternative semantics for $[\bar{x} \leftarrow \bar{p}]\, \phi$ is not a solution: this would invalidate the important proof rule (app) (Sect. 3), and in the case of the continuation monad would lead to an interpretation of $[x \leftarrow p]\, (\phi\ x)$ which constrains $p : TA = (A \to R) \to R$ only on those continuations $A \to R$ that factor through $\phi : A \to \Omega$.

Related Work
Global evaluation logic in the above sense is strongly related to Moggi's global semantics of evaluation logic [8] (although Moggi uses the term evaluation logic, the formulae denoted $[x \Leftarrow p]\,\phi$ in loc. cit. correspond essentially to $[\![x \leftarrow p]\!]\,\phi$ above, rather than to the modal formulae $[x \leftarrow p]\,\phi$ of evaluation logic in the original sense [12] or monad-based dynamic logic). In [8], it is assumed that T preserves monos, so that a predicate on A, represented as an admissible subobject $Q \hookrightarrow A$, lifts to a predicate TQ on TA, in order to define $[\![x \leftarrow p]\!]\,\phi$ as the formula

$$(\text{do } x \leftarrow p;\ \text{ret}\langle y, x\rangle) \in T\phi$$

(expressed for the sake of readability in the computational metalanguage). Here, $y : B$ is the context, so that ϕ is a predicate on $B \times A$. Roughly speaking, one can show that this semantics agrees with ours in case the base category has equality, i.e. diagonals are admissible subobjects, and T preserves admissible monos and their inverse images (so that T is simple [8]). In cases where these conditions fail, we have the impression that the semantics used here is slightly easier to handle (as well as more general as it does not require preservation of monos — preservation of monos may fail e.g. for the continuation monad over base categories other than **Set** [8]).

These remarks aside, the main difference between global evaluation logic and the logic considered in [8] is that the latter includes the full equational logic of the computational metalanguage of [7], that is, equality of programs is part of the logic. This fact plays a crucial technical role in the completeness proof presented in [8], where term models for a theory $\mathcal{T}$ are equipped with an equality determined by the equations derivable from $\mathcal{T}$. Contrastingly, global evaluation logic is restricted to formulae of the form $[\![x \leftarrow p]\!]\,\phi$, which talk only about results of programs rather than equalities between programs; thus, global evaluation logic provides a black-box view of programs, while the logic of [8] provides a glass-box view.

3 A Proof Calculus for Global Evaluation Logic

We now present a proof system for global evaluation logic *over simple monads* (finding a sound and complete calculus for the general case remains an open problem). The calculus is shown in Fig. 1, where Φ denotes the set of premises. Double lines indicate that a rule may be applied in both directions. The calculus is slightly modified w.r.t the one presented in [14]: the structural rules have been gathered in a slightly stronger rule (derivability of the original rules will be shown below), rules have been added to support the underlying equational theory, and a rule concerning equality under the global modality (which is not supported in the present framework, but could be added in later extensions) has been removed. Importantly, we allow rule (ctr) to be applied backwards, which is sound only for simple monads. We write $\Phi \vdash [\![\bar{x} \leftarrow \bar{p}]\!]\,\phi$ if $[\![\bar{x} \leftarrow \bar{p}]\!]\,\phi$ is derivable from a set Φ of formulae by means of the calculus. Instead of $\emptyset \vdash [\![\bar{x} \leftarrow \bar{p}]\!]\,\phi$, we write $\vdash [\![\bar{x} \leftarrow \bar{p}]\!]\,\phi$. Moreover, if $\Phi, [\![\bar{x} \leftarrow \bar{p}]\!]\,\phi \vdash [\![\bar{y} \leftarrow \bar{q}]\!]\,\psi$ and $\Phi, [\![\bar{y} \leftarrow \bar{q}]\!]\,\psi \vdash [\![\bar{x} \leftarrow \bar{p}]\!]\,\phi$, then we say that $[\bar{x} \leftarrow \bar{p}]\, \phi$ and $[\bar{y} \leftarrow \bar{q}]\, \psi$ are provably equivalent relative to Φ, and write $[\bar{x} \leftarrow \bar{p}]\, \phi \overset{\Phi}{\vdash\dashv} [\bar{y} \leftarrow \bar{q}]\, \psi$, again omitting the mention of Φ if Φ is empty.

$$(\top)\ \frac{\mathcal{E} \vdash \xi = \top}{[\bar{x} \leftarrow \bar{p}]\, \xi} \qquad \textbf{(cong)}\ \frac{\begin{array}{c}[\bar{x} \leftarrow \bar{p}]\, (\xi \leftrightarrow \chi) \\ [\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{r}[\xi/y]]\, \eta[\xi/y]\end{array}}{[\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{r}[\chi/y]]\, \eta[\chi/y]}$$

$$\textbf{(pre)}\ \frac{[\bar{y} \leftarrow \bar{q}]\, \phi}{[x \leftarrow p; \bar{y} \leftarrow \bar{q}]\, \phi}\, x \notin FV(\Phi) \qquad (\mathcal{E})\ \frac{\begin{array}{c}\mathcal{E} \vdash t = t' \\ [\ldots; x \leftarrow \operatorname{ret} t; \ldots]\, \eta\end{array}}{[\ldots; x \leftarrow \operatorname{ret} t'; \ldots]\, \eta}$$

$$(\eta)\ \begin{array}{c}[\bar{x} \leftarrow \bar{p}; y \leftarrow \operatorname{ret} r; \bar{z} \leftarrow \bar{q}]\, \eta \\ \hline\hline [\bar{x} \leftarrow \bar{p}; \bar{z} \leftarrow \bar{q}[r/y]]\, \eta[r/y]\end{array} \quad \textbf{(ctr)}\ \begin{array}{c}[\ldots; x \leftarrow p; y \leftarrow q; \bar{z} \leftarrow \bar{r}]\, \eta \\ \hline\hline [\ldots; y \leftarrow (\mathrm{do}\ x \leftarrow p;\ q); \bar{z} \leftarrow \bar{r}]\, \eta\end{array}\, x \notin FV(\eta, \bar{r})$$

Fig. 1. Proof calculus for global evaluation logic (over premises Φ)

Lemma 5. *The rules*

$$(\wedge\mathbf{I})\ \frac{\begin{array}{c}[\bar{x} \leftarrow \bar{p}]\, \phi \\ [\bar{x} \leftarrow \bar{p}]\, \psi\end{array}}{[\bar{x} \leftarrow \bar{p}]\, (\phi \wedge \psi)} \quad \textbf{(wk)}\ \frac{\begin{array}{c}\bar{x} \notin FV(\Phi) \\ \phi \to \psi \\ [\bar{x} \leftarrow \bar{p}]\, \phi\end{array}}{[\bar{x} \leftarrow \bar{p}]\, \psi} \quad \textbf{(app)}\ \frac{\begin{array}{c}[\bar{x} \leftarrow \bar{p}]\, \phi \\ y \notin FV(\phi)\end{array}}{[\bar{x} \leftarrow \bar{p}; y \leftarrow q]\, \phi}$$

are derivable (where Φ is again the set of premises).

Moreover, note that rules $(\mathcal{E})$ and (η) allow exchanging t for t' at any place in a formula whenever $\mathcal{E} \vdash t = t'$.

Remark 6. Under the additional restriction on signatures that an operation $f \in \Sigma$ has an argument of type Ω only if its result is of type Ω as well, rule (cong) can in fact be equivalently replaced by rules ($\wedge$I), (wk), and (app).

Lemma 7 (Substitution rule). *If σ is a substitution and $\Phi \vdash [\bar{x} \leftarrow \bar{p}]\, \phi$, then $\Phi\sigma \vdash ([\bar{x} \leftarrow \bar{p}]\, \phi)\sigma$, where application of σ is by capture-avoiding substitution of free variables.* □

Theorem 8 (Soundness for simple models). *Let T be a simple model such that $T \models \Phi$ for a set Φ of formulae. Then $\Phi \vdash [\bar{x} \leftarrow \bar{p}]\, \phi$ implies $T \models [\bar{x} \leftarrow \bar{p}]\, \phi$.*

4 Completeness

We will now prove strong completeness of global evaluation logic over simple monads. We explicitly fix the relevant notion of semantic consequence:

Definition 9. We write $\Phi \models [\bar{x} \leftarrow \bar{p}]\, \phi$ for a formula $[\bar{x} \leftarrow \bar{p}]\, \phi$ and a set Φ of formulae if for every simple model T, $T \models \Phi$ implies $T \models [\bar{x} \leftarrow \bar{p}]\, \phi$.

Theorem 10 (Strong completeness for simple models). *Let Φ be a set of formulae. Then $\Phi \models [\bar{x} \leftarrow \bar{p}]\, \phi$ implies $\Phi \vdash [\bar{x} \leftarrow \bar{p}]\, \phi$.*

The proof requires a number of lemmas, the most important of which is a normalization result for global evaluation formulae (Lemma 15).

Definition 11. A term is called *product β-normal* if it does not contain subterms of the form $\mathit{fst}(u, v)$ or $\mathit{snd}(u, v)$.

Lemma 12. *Every term t has a* product β-normal form*, i.e. a product β-normal term t' such that $\mathcal{E} \vdash t = t'$.*

Lemma 13. *Product β-normal terms of T-free type do not contain subterms of computational type.*

Proof. (Sketch) Induction over terms, using the restriction on argument types of operation symbols. □

Lemma 14. *Let t be product β-normal. Then there exists a decomposition $t = s[t_1/x_1, \dots t_n/x_n]$ such that*

– *s does not contain signature symbols and*
– *the t_i do not contain return or binding operators.*

Proof. (Sketch) Induction over the height of t, using Lemma 13 for the case that t is an application $f(q)$ of an operation symbol f. □

Lemma 15. *Given a formula $[\bar{x} \leftarrow \bar{p}]\, \phi$ and a variable $y : TA \in FV(\bar{p})$ there exists a formula $[\bar{z} \leftarrow \bar{q}]\, \psi$ provably equivalent to $[\bar{x} \leftarrow \bar{p}]\, \phi$, with $y \notin FV(\psi)$, and $y \in FV(q_j)$ only in case $q_j = y$.*

Proof. By successive reduction of $[\bar{x} \leftarrow \bar{p}]\, \phi$ to the required form. At the beginning of every step, reduce ϕ and the p_i to product β-normal form (Lemma 12), so that in particular $y \notin FV(\phi)$ by Lemma 13. Stop the process if none of the p_i contains return or binding. Otherwise, let p_i be the rightmost item containing return or binding. By Lemma 14, p_i is either a binding or an application of ret. Accordingly, perform at p_i a reduction step (thus following a rightmost strategy) to a provably equivalent formula using rule (ctr) in the 'up' direction or rule (η) in the 'down' direction, respectively, and continue.

At the end of this process, we have reached the required form $[\bar{z} \leftarrow \bar{q}]\, \psi$: This is clear if q_j is a variable. If q_j is of the form $e_1(\dots e_k(t) \dots)$, where $e_k \in \{\mathit{fst}, \mathit{snd}\}$, $k \geq 1$, with t not an application of a projection, then t is either a variable or an application $f(u)$ of an operation symbol f. In the former case, $t \not\equiv y$, since t is of product type, so that $y \notin FV(q_j)$. In the latter case, u is of T-free type by the restriction on signatures; by Lemma 14, $y \notin FV(q_j)$. The same applies if q_j is an application of an operation symbol. By product β-normality, no other cases occur.

It remains to prove termination. We introduce a well-founded strict ordering $\succ$ on term sequences and prove that the reductions are decreasing under $\succ$. Let $\mathit{Rets}(t)$ be the maximal depth of nested occurrences of the return operator in a term t. For a term sequence $\bar{p}$, let $\mu(\bar{p})$ be the multiset formed by the $\mathit{Rets}(p_i)$, and let $h(\bar{p})$ be the multiset of heights of the p_i, where we count sequential do-bindings $\mathrm{do}\ \bar{x} \leftarrow \bar{p};\ q$ as *one* application (rather than n applications) of do. We define the strict well-founded ordering $\gg$ on multisets of naturals as the multiset extension [1, 5] of the standard ordering $>$ of the naturals, generated by closure under multiset union and transitivity from the clauses

$$\{a\} \gg \{b_1, \dots, b_r\} \text{ if } a > b_i \text{ for all } i.$$

Then we put $\bar{p} \succ \bar{p}'$ iff $\langle \mu(\bar{p}), h(\bar{p})\rangle$ is lexicographically greater than $\langle \mu(\bar{p}'), h(\bar{p}')\rangle$. Upwards reduction by (ctr) at p_i is decreasing: the first component stays unchanged and the second decreases, as p_i is replaced by two terms of lesser height. For reduction by (η), recall that p_j does not contain ret for $j > i$, so that $\mathit{Rets}(p_j[q/x_i]) < \mathit{Rets}(\mathrm{ret}\, q)$. Hence $\mu(\bar{p}) \gg \mu(\bar{p}')$, and thus $\bar{p} \succ \bar{p}'$. Finally, the intermediate reductions to product β-normal form are height-decreasing, and thus decreasing w.r.t. $\succ$. □

Note that the previous proof essentially establishes a form of weak normalization for program sequences, for a rightmost reduction strategy.

As indicated above, a form of Leibniz equality will be imposed on the term model to be constructed below:

Definition 16. Let $t, s : TA$, and let Φ be a set of formulae. We say that t and s are *Leibniz equal* (under Φ), and write $t \sim_\Phi s$, if

$$[\![\bar{x} \leftarrow \bar{p}[t/y]]\!]\, \phi \overset{\Phi}{\dashv\vdash} [\![\bar{x} \leftarrow \bar{p}[s/y]]\!]\, \phi$$

for all $\phi : \Omega$ and all program sequences $\bar{p}$.

As a direct consequence of Lemma 15, we have

Lemma 17. *Let $t, s : TA$, and let Φ be a set of formulae. Then $t \sim_\Phi s$ iff t and s are* weakly Leibniz equal *(under Φ), i.e.*

$$[\![\bar{v} \leftarrow \bar{u}; x \leftarrow t; \bar{z} \leftarrow \bar{r}]\!]\, \phi \overset{\Phi}{\dashv\vdash} [\![\bar{v} \leftarrow \bar{u}; x \leftarrow s; \bar{z} \leftarrow \bar{r}]\!]\, \phi$$

for all $\phi : \Omega$ and all program sequences $\bar{v} \leftarrow \bar{u}$, $\bar{z} \leftarrow \bar{r}$.

This now allows us to prove the non-trivial direction ('only if') of what will turn out to be the truth lemma for our term model:

Lemma 18 (Truth lemma). *For every set Φ of formulae, $\Phi \vdash [\![\bar{y} \leftarrow \bar{q}]\!]\, \psi$ iff* $\mathrm{do}\ \bar{y} \leftarrow \bar{q};\ \mathrm{ret}\langle \bar{y}, \psi\rangle \sim_\Phi \mathrm{do}\ \bar{y} \leftarrow \bar{q};\ \mathrm{ret}\langle \bar{y}, \top\rangle$.

The proof of the completeness theorem is now by a term model construction which proceeds similarly as in [10]. Given an equational theory $\mathcal{E} = (\Sigma, E)$ and a set Φ of Σ-formulae, we construct a model of $\mathcal{E}$ as follows. Let $\mathbf{C}_{\mathcal{E},\Phi}$ be the category with objects being all types over Σ, and morphisms $A \to B$ being terms in context

$$(x : A \rhd t : B)$$

modulo Leibniz equality under Φ (we omit explicit notation for equivalence classes). It is easy to check that Leibniz equality is a congruence for the computational metalanguage. Identities in $\mathbf{C}_{\mathcal{E},\Phi}$ are given by variables $x : A \rhd x : A$, and composition is given by substitution. The category $\mathbf{C}_{\mathcal{E},\Phi}$ comes with a canonical cartesian structure, which on objects is just given by $A \times B$ and 1, with projections, pairing, and unique morphisms into 1 given in the obvious way. Axiom ($\mathcal{E}$) ensures that this does define a cartesian category satisfying $\mathcal{E}$.

Now we turn the type constructor T into a simple strong monad on $\mathbf{C}_{\mathcal{E},\Phi}$, with the unit η given by ret, the Kleisli star defined by

$$(x : A \rhd q : TB)^* := (p : TA \rhd \text{do } x \leftarrow p;\ q : TB),$$

and the strength given by

$$t_{A,B} := (p : A \times TB \rhd \text{do } x \leftarrow \mathit{snd}(p);\ \text{ret}\langle \mathit{fst}(p), x\rangle : T(A \times B)).$$

In the verification of the monadic laws and simplicity, crucial use is made of the fact that by Lemma 17, it suffices to prove weak Leibniz equality.

The completeness proof is finished by noting that the Truth Lemma states that $T \models [\bar{y} \leftarrow \bar{q}]\,\psi$ iff $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\,\psi$.

5 Conclusion

We have defined the syntax, semantics, and proof calculus of global evaluation logic over a simple monad on a cartesian base category with a distinguished Heyting algebra object. We have proved completeness of the logic using a term model in which, due to the absence of program equality from the logic, equality of terms is defined as Leibniz equality. The crucial point in the model construction is the reduction of Leibniz equality to a simplified form, which requires a normalization procedure on program sequences that uses a multiset termination measure. Two improvements of this result are left for future investigation: the extension of the calculus to sum types, with models over distributive categories, and support for user-defined control structures, i.e. basic operations with arguments of computational type.

The completeness of global evaluation logic forms the basis for investigations into the completeness of more advanced program logics defined in terms of it, in particular monad-based Hoare logic [13]. Moreover, global evaluation logic is expected to play a role in the design of a specification logic for the continuation monad: the latter has been shown not to admit an interpretation of dynamic logic [14], and Hoare logic over it lacks expressivity, while the much simpler global evaluation logic does make good sense over the continuation monad. As the continuation monad fails to be simple, this puts on the agenda the design of a complete calculus for the general case, without the simplicity assumption.

Acknowledgements. This work forms part of the DFG-funded project HasCASL (KR 1191/7-1 and KR 1191/7-2) and the project HOL-MDL funded by the FNK of the University of Bremen.

References

[1] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Comm. ACM*, 22:465–476, 1979.

[2] M. Fluet and G. Morrisett. Monadic regions. In *International Conference on Functional Programming*, pages 103–114. ACM, 2004.

[3] G. Hutton and E. Meijer. Monadic Parsing in Haskell. *J. Functional Programming*, 8:437–444, 1998.

[4] B. Jacobs and E. Poll. Coalgebras and Monads in the Semantics of Java. *Theoret. Comput. Sci.*, 291:329–349, 2003.

[5] J. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–117. Oxford, 1992.

[6] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1997.

[7] E. Moggi. Notions of computation and monads. *Inform. and Comput.*, 93:55–92, 1991.

[8] E. Moggi. A semantics for evaluation logic. *Fund. Inform.*, 22:117–152, 1995.

[9] E. Moggi and A. Sabry. Monadic encapsulation of effects: A revised approach (extended version). *J. Funct. Programming*, 11:591–627, 2001.

[10] T. Mossakowski, L. Schröder, and S. Goncharov. Completeness of monad-based dynamic logic. Technical report, University of Bremen, 2006.

[11] S. Peyton-Jones, editor. *Haskell 98 Language and Libraries — The Revised Report*. Cambridge, 2003. Also: J. Funct. Programming **13** (2003).

[12] A. Pitts. Evaluation logic. In *Higher Order Workshop*, Workshops in Computing, pages 162–189. Springer, 1991.

[13] L. Schröder and T. Mossakowski. Monad-independent Hoare logic in HasCasl. In *Fundamental Aspects of Software Engineering*, volume 2621 of *LNCS*, pages 261–277, 2003.

[14] L. Schröder and T. Mossakowski. Monad-independent dynamic logic in HasCasl. *J. Logic Comput.*, 14:571–619, 2004.

[15] M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoret. Comput. Sci.*, 342:28–55, 2005.

NOF-Multiparty Information Complexity Bounds for Pointer Jumping

Andre Gronemeier[⋆]

FB Informatik, LS2, Univ. Dortmund, 44221 Dortmund, Germany
andre.gronemeier@cs.uni-dortmund.de

Abstract. We prove a lower bound on the communication complexity of pointer jumping for multiparty one-way protocols in the number on the forehead model that satisfy a certain information theoretical restriction: We consider protocols for which the ith player may only reveal information about the first $i+1$ inputs. To this end we extend the information complexity approach of Chakrabarti, Shi, Wirth, and Yao (2001) and Bar-Yossef, Jayram, Kumar, and Sivakumar (2004) to our restricted version of the multiparty number on the forehead model. The best currently known multiparty protocol for pointer jumping by Damm, Jukna, and Sgall (1998) works in this model.

1 Introduction

1.1 Multiparty Communication Complexity

In the multiparty communication game by Chandra, Furst, and Lipton [8] k players jointly compute a function $f(x_1, \ldots, x_k)$ on k variables such that in the end each of the players knows the result. The players have unlimited computational power, but the ith player does not know the input variable x_i. Thus the players need to communicate to fulfill their task. This is usually called *number on the forehead model* (briefly NOF-model), since we can imagine the input x_i being written on the ith player's forehead. The players exchange messages according to a fixed protocol by writing to a shared blackboard seen by all players. Inscriptions on the blackboard are never deleted, each player appends his message to the previously written messages on the board. The current inscription on the blackboard determines unambiguously whose turn it is to write the next message and when to stop the protocol. The only important computational resource in this model is communication: The cost of the protocol is the worst case length of the inscription on the blackboard. The communication complexity of a function is the minimum cost of a protocol for the function.

Communication complexity for two players has been investigated independently [18] and is well understood [13,12]. Less is known about general multiparty protocols with more than two players. At the time of writing, only a single general proof method for proving lower bounds on the multiparty communication complexity of functions for more than two players is known: The discrepancy

[⋆] Supported by DFG grant SA 1053/1-1.

method by Babai, Nisan, and Szegedy [3] (see also [9,17]). Consequently, restricted versions of the multiparty model received some attention, most notably the simultaneous message model and the one-way model (see [1,13]).

In the simultaneous message model the players do not interact. Each player sends a single message, depending only on the inputs seen by the player, to a referee, who does not see the input. The referee has to announce the function value. Babai, Gál, Kimmel, and Lokam [1] introduced a proof method for proving lower bounds in the simultaneous message model which has found many applications by now, for example [16,6,4].

In the one-way model the interaction of the players is restricted such that the first player sends the first message message, then the second player sends a message depending on the inputs seen by him and the first message, and so on. The last player has to announce the function value. Although this model is still severely restricted compared to the general model in which the players can exchange messages in an arbitrary order, for more than three players currently no proof methods are known which make use of this restriction alone. The only known lower bound for a variable number of players in the one-way model by Damm, Jukna, and Sgall [11] uses an additional restriction that is described in Sect. 1.4.

1.2 Information Complexity

Information theory (see [10] for an introduction) has been used before to obtain results on communication complexity, but it has been used mainly as a tool in small parts of the proofs. Some references to these results are contained in [4]. Recently several publications emerged in which information theory is the main ingredient of the proof [7,4,5]. Bar-Yossef, Jayram, Kumar, and Sivakumar [4] reduced communication complexity problems to information theory problems and solved these problems in the information theory domain. Chakrabarti, Shi, Wirth, and Yao [7] introduced the concept of information complexity for the two party model. The information complexity of a function is the amount of information about the input that the messages of any protocol for the function must reveal. In the language of information theory, the information complexity of a function is the minimum mutual information between the messages of any protocol for the function and the inputs (see Sect. 2.2 and 2.3). In [5] this concept was further refined by Bar-Yossef, Jayram, Kumar, and Sivakumar.

1.3 Our Result

We consider the NOF-multiparty one-way model with an additional information theoretical restriction. In this model we prove a lower bound on the information complexity of the pointer jumping function.

Definition 1. *Let $f_1, \ldots, f_k$ be functions with domain and range $\{1, \ldots, n\}$. Then the k-player pointer jumping function* Jump_n^k *is defined as follows:*

$$\mathrm{Jump}_n^k(f_1, \ldots, f_k) := (f_k \circ f_{k-1} \circ \cdots \circ f_1)(1) \ .$$

Note that for the first input f_1 only $f_1(1)$ influences the result. The inputs $f_1(2), \dots, f_1(n)$ are redundant.

Our lower bound on the communication complexity of Jump_n^k holds for all one-way protocols for which the messages $M_1, \dots, M_{i-2}$ of the players $1, \dots, i-2$ do not reveal any information about the input f_i, in information theoretical terms, one-way protocols for which the mutual information between the input f_i and the messages $M_1, \dots, M_{i-2}$ is 0. We call protocols which obey this restriction *myopic* (see Sect. 2.3). The currently best one-way multiparty protocol for Jump_n^k by Damm, Jukna, and Sgall [11] is myopic.

We will prove the following lower bound on the multiparty communication complexity of Jump_n^k in the number on the forehead model.

Theorem 1. *Every myopic ε-error k-party one-way protocol in the number on the forehead model for* Jump_n^k *has cost $\Omega(n^{(1-\varepsilon)/k} \log n)$.*

This result is based on an information complexity bound and the proof relies solely on information theoretical arguments. Our result is not stronger than bounds which can be proved in the simultaneous message model and we need our additional information theoretical restriction, but the result extends the information complexity approach beyond the number in the hand model. To the best of our knowledge, this is the first extension of the information complexity results of Chakrabarti, Shi, Wirth, and Yao [7] and Bar-Yossef, Jayram, Kumar, and Sivakumar [5] to the one-way multiparty model.

1.4 Related Work

The communication complexity of pointer jumping has been investigated mainly for a two-player version that differs slightly from Def. 1. In this model upper and lower bounds have been proved by Nisan and Wigderson [14] and Ponzio, Radhakrishnan, and Venkatesh [15].

Much less is known about the communication complexity of pointer jumping, as defined in Def. 1, in the multiparty NOF-model. Wigderson proved an $\Omega(n^{1/2})$ bound for the complexity of pointer jumping for three players in the fully general one-way model (The result is contained in the appendix of [2]). The proof method of Babai *et al.* [1] yields an $\Omega(n^{1/k})$ lower bound on the communication complexity of Jump_n^k in the simultaneous message model [16]. The currently best one-way protocol for Jump_n by Damm, Jukna, and Sgall [11] has cost $O(n)$ for $k \geq \log^* n$ players and cost $n \log^{(k-1)} n + O(n)$ for $k < \log^* n$ players. In addition, Damm, Jukna, and Sgall proved an $\Omega(n/k^2)$ lower bound for up to $O(n^{1/3-\varepsilon})$ players in a restricted one-way model which they call *conservative one-way multiparty complexity*. In this model the ith player knows the inputs $f_{i+1}, \dots, f_k$, but unlike the usual number on the forehead model, instead of the inputs $f_1, \dots, f_{i-1}$, he does only know the partial result $(f_{i-1} \circ f_{i-2} \circ \dots \circ f_1)(1)$. Since this result can take only n different values whereas the inputs $f_1, \dots, f_{i-1}$ can take $n^{(i-1)n}$ different values, this is a potentially severe restriction.

Note that the above result is complementary to our result in the following sense: In a myopic protocol for Jump_n^k the ith player must not reveal information about $f_{i+2}, \dots, f_k$. This is obviously the case if his message does not depend functionally on these inputs. Thus myopic protocols include protocols in which the ith player may only access the inputs f_j with $j < i$ and, in addition, the input f_{i+1}. For conservative protocols, the ith player has unrestricted access to the inputs f_j with $j > i$, while the access to the inputs f_j with $j < i$ is severely restricted. Note that a protocol can be conservative and myopic at the same time. Although this is a very severe restriction, the currently best one-way protocol for Jump_n^k of Damm, Jukna, and Sgall [11] is both conservative and myopic.

2 Preliminaries

2.1 Notation

We use $[n]$ as an abbreviation for the set $\{1, \dots, n\}$. Let P be a k-party one-way protocol for Jump_n^k. If the inputs of P are drawn randomly with respect to some probability distribution, then these inputs and the messages are random variables $F_1, \dots, F_k$ and $M_1, \dots, M_{k-1}$, respectively. In this case let $\mathcal{F}$ denote the set of the random variables $\{F_1, \dots, F_k\}$ and let $\mathcal{M}_i$ denote the set of the random variables $\{M_1, \dots, M_i\}$. Furthermore let $\widetilde{F}_i := F_i \circ F_{i-1} \circ \dots \circ F_1$, hence $\text{Jump}_n^k(F_1, \dots, F_k) = \widetilde{F}_k(1)$.

2.2 Tools from Information Theory

In this section some basic facts from information theory are summarized. Results that are needed for an arbitrary number of random variables are only stated for two variables. In most cases the extension to an arbitrary number of variables follows immediately by induction. Most information theoretical facts that we use are elementary. Nevertheless, we see this section merely as an agreement on the notation of information theoretical results. For a proper introduction to information theory we refer the reader to the book by Cover and Thomas [10].

Let X, Y, Z, and W be random variables with a finite range R. Then $\text{H}(X) := \sum_{x \in R} \text{Prob}(X = x) \log(1/\text{Prob}(X = x))$ is called the *entropy* of X, $\text{H}(X, Y)$ is the entropy of the joint distribution of X and Y, and $\text{H}(X \mid Y) := \text{H}(X, Y) - \text{H}(Y)$ is called the conditional entropy of X given Y. The *mutual information* between X and Y is defined as $\text{I}(X; Y) := \text{H}(X) - \text{H}(X \mid Y)$ and the *conditional mutual information* between X and Y given Z is $\text{I}(X; Y \mid Z) := \text{H}(X \mid Z) - \text{H}(X \mid Y, Z)$.

Let E denote an event, for example $W = w$. Then $\text{H}(X \mid E)$ denotes the entropy of X with respect to the conditional distribution of X given the event E occurred. Conditioning on an event for conditional entropy, mutual information and conditional mutual information is defined analogously. For example, $\text{I}(X; Y \mid Z, W{=}w)$ is the mutual information between X and Y given Z with respect to the conditional distribution given the event $W{=}w$ occurred.

Proofs of the following elementary properties of entropy and mutual information can be found in most textbooks about information theory, for example [10].

Theorem 2. *Let X, Y, Z, and W be random variables with finite range R. Then*

1. $0 \leq \mathrm{H}(X) \leq \log|R|$ *with* $\mathrm{H}(X) = \log|R|$ *iff X is uniformly distributed.*
2. $\mathrm{H}(X \mid Y) = \sum_{y \in R} \mathrm{Prob}(Y{=}y)\ \mathrm{H}(X \mid Y{=}y)$.
3. $\mathrm{I}(X;Y \mid Z) = \sum_{z \in R} \mathrm{Prob}(Z{=}z)\ \mathrm{I}(X;Y \mid Z{=}z)$.
4. $\mathrm{I}(X;Y \mid Z,W) = \sum_{w \in R} \mathrm{Prob}(W{=}w)\ \mathrm{I}(X;Y \mid Z,W{=}w)$.
5. $\mathrm{H}(X,Y) \leq \mathrm{H}(X) + \mathrm{H}(Y)$ *with equality iff X and Y are independent.*
6. *If X and Y are jointly independent of Z then* $\mathrm{H}(X \mid Y,Z) = \mathrm{H}(X|Y)$.
7. *If X and Y are jointly independent of Z then* $\mathrm{I}(X;Y \mid Z) = \mathrm{I}(X;Y)$.
8. *If f is a function with domain R then* $\mathrm{H}(X, f(X)) = \mathrm{H}(X)$.

The following useful inequality can be proved easily using Jensen's inequality.

Lemma 1. *Let $a_1, \ldots, a_n \in \mathbb{R}$ and $\mu : [n] \longrightarrow [0,1]$ be a probability distribution on $[n]$. Then*

$$\sum_{i=1}^{n} \mu(i) \log a_i \leq \log\left(\sum_{i=1}^{n} \mu(i) a_i\right) \ .$$

Fano's inequality uses information theory to give bounds on the error of a predictor.

Theorem 3 (Fano's inequality). *Let X and Y be random variables with range R_X and R_Y, let $P : R_Y \longrightarrow R_X$ be a function that predicts the value of X from an observed value of Y, and let $\varepsilon = \mathrm{Prob}(P(Y) \neq X)$ be the prediction error. Then*

$$\mathrm{H}_2(\varepsilon) + \varepsilon \log(|R_X| - 1) \geq \mathrm{H}(X \mid Y)$$

where $\mathrm{H}_2(\varepsilon) = \varepsilon \log(1/\varepsilon) + (1-\varepsilon)\log(1/(1-\varepsilon))$ *denotes the binary entropy function.*

Note that Fano's inequality implies $1 + \varepsilon \log(|R_X| - 1) \geq \mathrm{H}(X \mid Y)$ since the binary entropy function is bounded from above by 1.

2.3 Communication Complexity and Information Complexity

The multiparty communication game by Chandra, Furst, and Lipton [8] and multiparty one-way protocols [1,13] have been described already in the introduction. The following definition summarizes the introductory discussion.

Definition 2. *In a k-party one-way protocol P with input variables $x_1, \ldots, x_k$ the ith player sees all input variables except x_i. Each player $i \in \{1, \ldots, k-1\}$ sends a single message m_i which may depend on the inputs seen by player i and the messages $m_1, \ldots, m_{i-1}$ of the previous players. The kth player announces the output $P(x_1, \ldots, x_k)$ of the protocol depending on the inputs seen by the kth player and the messages $m_1, \ldots, m_{k-1}$.*

Let $f(x_1, \ldots, x_k)$ be a function on k variables and let μ be a distribution on the domain of f. The protocol P is called an ε-error protocol for f with respect to μ, if $\mathrm{Prob}_\mu(P(x_1, \ldots, x_k) \neq f(x_1, \ldots, x_k)) \leq \varepsilon$.

The cost $c(P)$ of the one-way k-party protocol P for f is the length of the longest message sent by any of the players. The ε-error one-way multiparty communication complexity $\mathrm{C}^1_{\mu,\varepsilon}(f)$ of the function f with respect to distribution μ is the minimum cost of an ε-error one-way k-party protocol for f. We omit μ, if μ is the uniform distribution.

Note that our definition of the cost of a one-way protocol differs from the usual definition from [8]. We define the cost of a protocol as the length of the longest message sent by any of the players, while usually the worst case length of the whole transcript of the communication is used. For k players these two cost measures can differ at most by a factor of k.

We will impose an additional information theoretical restriction on one-way protocols. In a *myopic* protocol with random inputs $X_1, \ldots, X_k$ the messages of the players $1, \ldots, i-2$ must not reveal any information about X_i.

Definition 3. *Let $X_1, \ldots, X_k$ be the inputs and $M_1, \ldots, M_{k-1}$ be the messages of a k-party one-way protocol P for f. Let $\mathcal{X}_i = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_k\}$. Then P is called myopic, if*

$$\mathrm{I}(X_i; M_1, \ldots, M_{i-2} \mid \mathcal{X}_i) = 0 \quad \textit{for all } 1 \leq i \leq k \ .$$

Let $\mathrm{C}^{\mathrm{m}}_{\mu,\varepsilon}(f)$ denote the minimum cost of a myopic ε-error one-way k-party protocol for f w.r.t. μ. We omit μ, if μ is the uniform distribution.

Note that for myopic protocols with independent inputs $X_1, \ldots, X_k$ the input X_i is independent of the messages $M_1, \ldots, M_{i-2}$, since

$$\mathrm{I}(X_i; M_1, \ldots, M_{i-2} \mid \mathcal{X}_i) = \mathrm{H}(X_i \mid \mathcal{X}_i) - \mathrm{H}(X_i \mid M_1, \ldots, M_{i-2}, \mathcal{X}_i) = 0$$

and therefore

$$\mathrm{H}(X_i) = \mathrm{H}(X_i \mid \mathcal{X}_i) = \mathrm{H}(X_i \mid M_1, \ldots, M_{i-2}, \mathcal{X}_i) \ .$$

The following lemma generalizes the information complexity approach of [7] and [5] to multiparty protocols. There are several sensible definitions of information complexity in the multiparty model. Therefore, instead of defining information complexity explicitly, we only state a lower bound on communication complexity in terms of mutual information that is meaningful for our application to myopic protocols.

Lemma 2. *Let f be a function on k random variables $X_1, \ldots, X_k$ that are jointly distributed with respect to the distribution μ and let $M_1, \ldots, M_{k-1}$ be the messages of a k-party one-way protocol P that computes f with error ε with respect to distribution μ. Then the cost of P is bounded from below by*

$$\max_{i \in [k-1]} \mathrm{I}(M_i; X_{i+1} \mid \mathcal{X}_{i+1}, \mathcal{M}_{i-1}) \ .$$

Proof. Let $|M_i|$ denote the number of different values that the random variable M_i can take. Clearly, the cost $c(P)$ of P is bounded by $\max_{i \in [k-1]} \log |M_i|$ and, since conditioning reduces entropy, the definition of conditional mutual information implies

$$c(P) \geq \max_{i\in[k-1]} \log|M_i| \geq \max_{i\in[k-1]} \mathrm{H}(M_i) \geq \max_{i\in[k-1]} \mathrm{I}(M_i; X_{i+1} \mid \mathcal{X}_{i+1}, \mathcal{M}_{i-1}) \ . \square$$

3 Main Result

3.1 Outline of the Proof

Consider the situation of the ith player in a myopic k-party protocol P for Jump_n^k with uniformly distributed inputs: The ith player knows the inputs $\mathcal{F} \setminus \{F_i\}$, and when it is his turn to send a message, he additionally knows the messages $\mathcal{M}_{i-1} = \{M_1, \ldots, M_{i-1}\}$ of the previous players. Since P is myopic, F_{i+1} is independent of the messages $\mathcal{M}_{i-1}$ and, in particular, $F_{i+1}(1), \ldots, F_{i+1}(n)$ are independent with respect to the conditional distribution given the first $i-1$ messages $\mathcal{M}_{i-1}$.

We will show in Lemma 4 that under these circumstances the message M_i can not reveal much information about $\widetilde{F}_{i+1}(1)$, if the conditional entropy of $\widetilde{F}_i(1)$ given $\mathcal{F}\setminus\{F_i\}$ and $M_1, \ldots, M_{i-1}$ is large and the conditional mutual information between M_i and F_{i+1} given $\mathcal{F} \setminus \{F_{i+1}\}$ and $M_1, \ldots, M_{i-1}$ is small. This claim can be used inductively to prove a lower bound on the conditional entropy of $\widetilde{F}_k(1)$ given $\mathcal{F} \setminus \{F_k\}$ and $M_1, \ldots, M_{k-1}$, if the conditional mutual information between M_i and F_{i+1} is bounded appropriately from above for all $i \in [k-1]$.

Intuitively, the last claim holds because the ith player needs to allocate the information about F_{i+1} to $F_{i+1}(1), \ldots, F_{i+1}(n)$, since these variables are independent, whereas only one of the variables, namely $F_{i+1}(\widetilde{F}_i(1))$, contains information about $\widetilde{F}_{i+1}(1)$. It is difficult for the ith player to predict the value of $\widetilde{F}_i(1)$, if the conditional entropy of $\widetilde{F}_i(1)$ given $\mathcal{F} \setminus \{F_i\}$ and $M_1, \ldots, M_{i-1}$ is large. Therefore he has to send a lot of useless information, if he wants to reveal some information about $F_{i+1}(\widetilde{F}_i(1))$. The details of this argument are contained in Lemma 3.

Finally, in Theorem 4 we will use Fano's inequality and Lemma 4 to prove a lower bound the cost of myopic protocols for Jump_n^k.

3.2 Proof of the Main Result

First we will show that a random variable M must contain much information about the n independent random variables $X = (X_1, \ldots, X_n)$, if it contains much information about a randomly chosen variable from this collection which is chosen independently of X and M with respect to a distribution with large entropy.

Lemma 3. *Let $X = (X_1, \ldots, X_n)$ be n independent random variables with $\mathrm{H}(X_p) \leq \log n$ for all p, let Y and M be random variables such that X and M are jointly independent of Y and let P be a function that maps Y to $[n]$. If $A := \lceil \mathrm{I}(X_1, \ldots, X_n; M \mid Y)/\log n \rceil < n/2$ then*

$$\mathrm{I}(X_{P(Y)}; M \mid Y) \le \frac{\log(n-A) + 1 - \mathrm{H}(P(Y))}{\log(n-A) - \log A} \log n \ .$$

Proof. Clearly $\mathrm{I}(X; M \mid Y) = \mathrm{I}(X; M)$ since X and M are jointly independent of Y. Since the variables $X_1, \ldots, X_n$ are independent, $\mathrm{H}(X_1, \ldots, X_n) = \sum_{p=1}^{n} \mathrm{H}(X_p)$, and obviously $\mathrm{H}(X_1, \ldots, X_n \mid M) \le \sum_{p=1}^{n} \mathrm{H}(X_p \mid M)$. Therefore, by using the definition of mutual information, it follows that

$$\mathrm{I}(X_1, \ldots, X_n; M \mid Y) = \mathrm{I}(X_1, \ldots, X_n; M) \ge \sum_{p=1}^{n} \mathrm{I}(X_p; M) \ .$$

Furthermore $\mathrm{I}(X_{P(Y)}; M \mid Y) = \mathrm{I}(X_{P(Y)}; M \mid Y, P(Y))$ since $P(Y)$ is a function of Y and $\mathrm{I}(X_{P(Y)}; M \mid Y, P(Y) = p) = \mathrm{I}(X_p; M)$ since X and M are jointly independent of Y. Therefore

$$\begin{aligned}\mathrm{I}(X_{P(Y)}; M \mid Y) &= \mathrm{I}(X_{P(Y)}; M \mid Y, P(Y)) \\ &= \sum_{p=1}^{n} \mathrm{Prob}(P(Y) = p) \cdot \mathrm{I}(X_{P(Y)}; M \mid Y, P(Y) = p) \\ &= \sum_{p=1}^{n} \mathrm{Prob}(P(Y) = p) \cdot \mathrm{I}(X_p; M) \ .\end{aligned}$$

Assume w.l.o.g. that $\mathrm{Prob}(P(Y){=}1) \ge \mathrm{Prob}(P(Y){=}2) \ge \cdots \ge \mathrm{Prob}(P(Y){=}n)$. Then the last sum is maximized, if $\mathrm{I}(X_p; M)$ is large for small values of p. Since $\mathrm{I}(X_p; M) \le \mathrm{H}(X_p) \le \log n$ and $\sum_{p=1}^{n} \mathrm{I}(X_p; M) \le \mathrm{I}(X_1, \ldots, X_n; M) \le A \cdot \log n$, we get an upper bound for the sum, if we assume that $\mathrm{I}(X_p, M) = \log n$ for $p \le A$ and $\mathrm{I}(X_p, M) = 0$ for $p > A$. Let Z be a random variable such that $Z = 1$ if $P(Y) \le A$ and $Z = 0$ if $P(Y) > A$. Then, using the upper bound described above, we get

$$\mathrm{I}(X_{P(Y)}; M \mid Y) \le \mathrm{Prob}(Z = 1) \cdot \log n \ .$$

The value of Z is a function of $P(Y)$. Hence $\mathrm{H}(P(Y)) = \mathrm{H}(P(Y), Z) = \mathrm{H}(Z) + \mathrm{H}(P(Y) \mid Z)$ and

$$\mathrm{H}(P(Y) \mid Z) = \mathrm{H}(P(Y)) - \mathrm{H}(Z) \ge \mathrm{H}(P(Y)) - 1 \ .$$

On the other hand

$$\begin{aligned}\mathrm{H}(P(Y) \mid Z) = {} & \mathrm{Prob}(Z = 1) \cdot \mathrm{H}(P(Y) \mid Z = 1) \\ & + (1 - \mathrm{Prob}(Z = 1)) \cdot \mathrm{H}(P(Y) \mid Z = 0) \\ \le {} & \mathrm{Prob}(Z = 1) \cdot \log A + (1 - \mathrm{Prob}(Z = 1)) \cdot \log(n - A)\end{aligned}$$

where the last inequality is due to the fact, that under the condition $Z = 1$ the values of $P(Y)$ are from $\{1, \ldots, A\}$ while under the condition $Z = 0$ the values of $P(Y)$ are from $\{A + 1, \ldots, n\}$. By combining the two inequalities we get

$$\mathrm{Prob}(Z = 1)[\log(n - A) - \log A] \le \log(n - A) + 1 - H(P(Y))$$

and the using the premise $A < n/2 \Leftrightarrow \log(n-A) - \log A > 0$ we get

$$\mathrm{Prob}(Z=1) \leq \frac{\log(n-A) + 1 - H(P(Y))}{\log(n-A) - \log A} \ .$$

Finally, by substituting this bound into our estimate of $\mathrm{I}(X_{P(Y)}; M \mid Y)$, we get the claimed result

$$\begin{aligned}\mathrm{I}(X_{P(Y)}; M \mid Y) &\leq \mathrm{Prob}(Z=1) \cdot \log n \\ &\leq \frac{\log(n-A) + 1 - H(P(Y))}{\log(n-A) - \log A} \log n \ . \qquad \square\end{aligned}$$

Now we will apply the last lemma to myopic one-way protocols for pointer jumping.

Lemma 4. *Let $M_1, \ldots, M_{k-1}$ be the messages of a myopic k-player one-way protocol P for* Jump_n^k *with uniformly distributed inputs $F_1, \ldots, F_k$ such that the cost of P satisfies $\lceil c(P)/\log n \rceil < n/2$ and the messages of P satisfy*

$$\mathrm{I}(F_{i+1}; M_i \mid \mathcal{F} \setminus \{F_{i+1}\}, \mathcal{M}_{i-1}) / \log n \leq C$$

for all $i < k$. Then

$$\mathrm{H}(\widetilde{F}_i(1) \mid \mathcal{F} \setminus \{F_i\}, \mathcal{M}_{i-1}) \geq \log n - i - i \log(C+1)$$

for all $i \leq k$.

Proof. Clearly, by the definition of mutual information,

$$\begin{aligned}&\mathrm{I}(\widetilde{F}_{i+1}(1); M_i \mid \mathcal{F} \setminus \{F_{i+1}\}, \mathcal{M}_{i-1}) = \\ &\qquad \mathrm{H}(\widetilde{F}_{i+1}(1) \mid \mathcal{F} \setminus \{F_{i+1}\}, \mathcal{M}_{i-1}) - \mathrm{H}(\widetilde{F}_{i+1}(1) \mid \mathcal{F} \setminus \{F_{i+1}\}, \mathcal{M}_{i}) \ .\end{aligned}$$

Since P is myopic, the first term on the right side of the last equation is equal to $\log n$. Let $B_i := \mathrm{H}(\widetilde{F}_i(1) \mid \mathcal{F} \setminus \{F_i\}, \mathcal{M}_{i-1})$. Then we get

$$B_{i+1} = \log n - \mathrm{I}(\widetilde{F}_{i+1}(1); M_i \mid \mathcal{F} \setminus \{F_{i+1}\}, \mathcal{M}_{i-1}) \ .$$

The message M_i does only depend on $\mathcal{F} \setminus \{F_i\}$ and $\mathcal{M}_{i-1}$. For fixed values of n and k one can easily show that there is only a finite number of different messages. Thus we can assume that $M_i \in \{0,1\}^m$ for some fixed m and use the messages of the protocol as an index of summation without worrying about convergence. For fixed f and m let $E_i(f,m)$ denote the event that $(F_1, \ldots, F_{i-1}, F_{i+2}, \ldots, F_k) = f$ and $(M_1, \ldots, M_{i-1}) = m$ (note that both F_i *and* F_{i+1} are not fixed). Then, by using $\widetilde{F}_{i+1}(1) = F_{i+1}(\widetilde{F}_i(1))$ and expanding the conditional mutual information, we get

$$B_{i+1} = \sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot \Big[\log n - \mathrm{I}(F_{i+1}(\widetilde{F}_i(1)); M_i \mid F_i, E_i(f,m))\Big] \ .$$

Under the condition $E_i(f,m)$ the messages $M_1,\ldots,M_{i-1}$ and all inputs except F_i and F_{i+1} are fixed to constants. In this case $\widetilde{F}_i(1)$ is a function of F_i, since $\widetilde{F}_i(1) = F_i(\widetilde{F}_{i-1}(1))$ and $\widetilde{F}_{i-1}(1)$ is constant under the condition $E_i(f,m)$. Similarly M_i is a function of F_{i+1}, the only variable seen by player i that is not fixed by the conditioning event. Furthermore, since P is myopic, F_{i+1} is independent of $M_1,\ldots,M_{i-1}$ and $\mathcal{F}\backslash\{F_{i+1}\}$. Thus $\mathrm{H}(F_{i+1} \mid F_i, E_i(f,m)) = \mathrm{H}(F_{i+1} \mid E_i(f,m))$ implying that F_{i+1} is independent of F_i under the condition $E_i(f,m)$. Hence under the condition $E_i(f,m)$ the random variables F_{i+1} and M_i are jointly independent of F_i and the random variables $F_{i+1}(p)$ for $p = 1,\ldots,n$ are independent and satisfy $\mathrm{H}(F_{i+1}(p) \mid E_i(f,m)) \le \log n$. Clearly even under the condition $E_i(f,m)$ the entropy of M_i is bounded from above by the cost of P, thus $\lceil \mathrm{I}(F_{i+1}; M_i \mid F_i, E_i(f,m))/\log n\rceil \le \lceil c(P)/\log n\rceil < n/2$. Therefore we can estimate $S_i(f,m) := \log n - \mathrm{I}(F_{i+1}(\widetilde{F}_i(1)); M_i \mid F_i, E_i(f,m))$ using Lemma 3 with $X_p = F_{i+1}(p)$, $Y = F_i$, $P(F_i) = F_i(\widetilde{F}_{i-1}(1)) = \widetilde{F}_i(1)$, $M = M_i$, and $A_{i+1}(f,m) := \lceil \mathrm{I}(F_{i+1}; M_i \mid F_i, E_i(f,m))/\log n\rceil$ to get

$$\begin{aligned}
S_i(f,m) &= \log n - \mathrm{I}(F_{i+1}(\widetilde{F}_i(1)); M_i \mid F_i, E_i(f,m))\\
&\ge \log n - \frac{\log(n - A_{i+1}(f,m)) + 1 - \mathrm{H}(\widetilde{F}_i(1) \mid E_i(f,m))}{\log(n - A_{i+1}(f,m)) - \log A_{i+1}(f,m)} \cdot \log n\\
&= \frac{\mathrm{H}(\widetilde{F}_i(1) \mid E_i(f,m)) - \log A_{i+1}(f,m) - 1}{\log(n - A_{i+1}(f,m)) - \log A_{i+1}(f,m)} \cdot \log n\\
&\ge \mathrm{H}(\widetilde{F}_i(1) \mid E_i(f,m)) - \log A_{i+1}(f,m) - 1 \enspace .
\end{aligned}$$

For the last inequality we use that $\log(n - A_{i+1}(f,m)) - \log A_{i+1}(f,m) \le \log n$. From this estimate of $S_i(f,m)$ we get

$$\begin{aligned}
B_{i+1} &= \sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot S_i(f,m)\\
&\ge \sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot \mathrm{H}(\widetilde{F}_i(1) \mid E_i(f,m))\\
&\quad - \sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot \log A_{i+1}(f,m) - 1 \enspace .
\end{aligned}$$

Let T_1 and T_2 denote the first and second term of the right hand side in the last inequality, respectively. Then

$$T_1 = \mathrm{H}(\widetilde{F}_i(1) \mid \mathcal{F} \setminus \{F_i, F_{i+1}\}, \mathcal{M}_{i-1}) \ge B_i$$

where the last inequality holds, because conditioning reduces entropy. We apply Lemma 1 to the second term and get

$$\begin{aligned}
T_2 &= \sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot \log A_{i+1}(f,m) \\
&\le \log \left(\sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot A_{i+1}(f,m) \right) \\
&= \log \left(\sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot \lceil \mathrm{I}(F_{i+1}; M_i \mid F_i, E_i(f,m)) / \log n \rceil \right) \\
&\le \log \left(\sum_{f,m} \mathrm{Prob}(E_i(f,m)) \cdot (\mathrm{I}(F_{i+1}; M_i \mid F_i, E_i(f,m)) / \log n + 1) \right) \\
&= \log \left(\mathrm{I}(F_{i+1}; M_i \mid \mathcal{F} \setminus \{F_{i+1}\}, \mathcal{M}_{i-1}) / \log n + 1 \right) \\
&\le \log(C+1) \ .
\end{aligned}$$

Thus $B_{i+1} \ge T_1 - T_2 - 1 \ge B_i - \log(C+1) - 1$ and the claim of the Theorem is implied by this recurrence relation and the base case $B_1 = \log n$. □

The main result follows from the last lemma by a simple application of Fano's inequality.

Theorem 4. *$\mathrm{C}^{\mathrm{m}}_{\varepsilon}(\mathrm{Jump}^k_n) \ge (2^{-(1+1/k)} n^{(1-\varepsilon)/k} - 2) \log n$.*

Proof. Let P be a myopic ε-error protocol for Jump^k_n, let $F_1, \dots, F_k$ be the random inputs of Jump^k_n, and let $M_1, \dots, M_{k-1}$ be the messages of P for this input. The kth player uses $F_1, \dots, F_{k-1}$ and $M_1, \dots, M_{k-1}$ to predict the value of $\mathrm{Jump}^k_n(F_1, \dots, F_k) = \widetilde{F}_k(1)$ with error ε, thus, by Fano's inequality,

$$1 + \varepsilon \log n \ge \mathrm{H}(\widetilde{F}_k(1) \mid \mathcal{F} \setminus \{F_k\}, \mathcal{M}_{k-1}) \ .$$

Recall that $c(P)$ denotes the cost of P and let $C := \lceil c(P)/\log n \rceil$. If $C \ge n/2$ then the claim of the Theorem holds for $k \ge 2$ and sufficiently large n. If $C < n/2$ then, by Lemma 2, $\lceil \mathrm{I}(F_{i+1}; M_i \mid \mathcal{F} \setminus \{F_{i+1}\}, \mathcal{M}_{i-1}) / \log n \rceil \le C$ for all $i \in [k-1]$, and consequently, by Lemma 4,

$$\mathrm{H}(\widetilde{F}_k(1) \mid \mathcal{F} \setminus \{F_k\}, \mathcal{M}_{k-1}) \ge \log n - k - k \log(C+1) \ .$$

Combining these inequalities yields $1 + \varepsilon \log n \ge \log n - k - k \log(C+1)$ which implies $C \ge n^{(1-\varepsilon)/k} / 2^{1+1/k} - 1$. The claim of the theorem follows immediately from the last inequality. □

Acknowledgment

Thanks to Martin Sauerhoff for proofreading and advice.

References

1. Babai, L., Gál, A., Kimmel, P.G., Lokam, S.V.: Communication complexity of simultaneous messages. SIAM J. Comput. **33** (2004) 137–166
2. Babai, L., Hayes, T.P., Kimmel, P.G.: The cost of the missing bit: Communication complexity with help. Combinatorica **21** (2001) 455–488
3. Babai, L., Nisan, N., Szegedy, M.: Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. J. Comput. Syst. Sci. **45** (1992) 204–232
4. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: Information theory methods in communication complexity. In: Proc. of 17th CCC. (2002) 93–102
5. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. J. Comput. Syst. Sci. **68** (2004) 702–732
6. Beame, P., Pitassi, T., Segerlind, N., Wigderson, A.: A direct sum theorem for corruption and the multiparty NOF communication complexity of set disjointness. In: Proc. of 20th CCC. (2005) 52–66
7. Chakrabarti, A., Shi, Y., Wirth, A., Yao, A.C.: Informational complexity and the direct sum problem for simultaneous message complexity. In: Proc. of 42nd FOCS. (2001) 270–278
8. Chandra, A.K., Furst, M.L., Lipton, R.J.: Multi-party protocols. In: Proc. of 15th STOC. (1983) 94–99
9. Chung, F.R.K., Tetali, P.: Communication complexity and quasi randomness. SIAM J. Discret. Math. **6** (1993) 110–125
10. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley-Interscience (1991)
11. Damm, C., Jukna, S., Sgall, J.: Some bounds on multiparty communication complexity of pointer jumping. Comput. Complex. **7** (1998) 109–127
12. Hromkovič, J.: Communication Complexity and Parallel Computing. Springer (2002)
13. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press (1997)
14. Nisan, N., Wigderson, A.: Rounds in communication complexity revisited. SIAM J. Comput. **22** (1993) 211–219
15. Ponzio, S., Radhakrishnan, J., Venkatesh, S.: The communication complexity of pointer chasing. J. Comput. Syst. Sci. **62** (2001) 323–355
16. Pudlák, P., Rödl, V., Sgall, J.: Boolean circuits, tensor ranks, and communication complexity. SIAM J. Comput. **26** (1997) 605–633
17. Raz, R.: The BNS-Chung criterion for multi-party communication complexity. Comput. Complex. **9** (2000) 113–122
18. Yao, A.C.: Some complexity questions related to distributive computing (preliminary report). In: Proc. of 11th STOC. (1979) 209–213

Dimension Characterizations of Complexity Classes

Xiaoyang Gu[⋆] and Jack H. Lutz[⋆⋆]

Department of Computer Science, Iowa State University, Ames, IA 50011 USA
{xiaoyang, lutz}@cs.iastate.edu

Abstract. We use derandomization to show that sequences of positive pspace-dimension – in fact, even positive Δ^{p}_k-dimension for suitable k – have, for many purposes, the full power of random oracles. For example, we show that, if S is any binary sequence whose Δ^{p}_3-dimension is positive, then $\mathrm{BPP} \subseteq \mathrm{P}^S$ and, moreover, every BPP promise problem is P^S-separable. We prove analogous results at higher levels of the polynomial-time hierarchy.

The *dimension-almost-class* of a complexity class $\mathcal{C}$, denoted by dimalmost- $\mathcal{C}$, is the class consisting of all problems A such that $A \in \mathcal{C}^S$ for all but a Hausdorff dimension 0 set of oracles S. Our results yield several characterizations of complexity classes, such as BPP = dimalmost-P and AM = dimalmost-NP, that refine previously known results on almost-classes. They also yield results, such as Promise-BPP = almost-P-Sep = dimalmost-P-Sep, in which even the almost-class appears to be a new characterization.

1 Introduction

Assessing the computational power of randomness is one of the most fundamental challenges facing computational complexity theory. Concrete questions involving the best algorithms for primality testing, factoring, etc., are instances of this challenge, as are structural questions concerning BPP, AM, and other randomized complexity classes.

One approach to studying the power of a randomized complexity class $\mathcal{C}$ is to address the following question: If $\mathcal{C}_0$ is the nonrandomized version of $\mathcal{C}$, then how weak an assumption can we place on an oracle S and still be assured that $\mathcal{C} \subseteq \mathcal{C}_0^S$? For example, how weak an assumption can we place on an oracle S and still be assured that $\mathrm{BPP} \subseteq \mathrm{P}^S$? For this particular question, it was a result of folklore that $\mathrm{BPP} \subseteq \mathrm{P}^S$ holds for every oracle S that is algorithmically random in the sense of Martin-Löf [21]; it was shown by Lutz [18] that $\mathrm{BPP} \subseteq \mathrm{P}^S$ holds for every oracle S that is pspace-random; and it was shown by Allender and Strauss [3] that $\mathrm{BPP} \subseteq \mathrm{P}^S$ holds for every oracle S that is p-random, or even random relative to a particular sublinear-time complexity class.

⋆ Research supported in part by National Science Foundation Grant 0344187.

⋆⋆ Part of the second author's work was carried out during a sabbatical at the University of Wisconsin.

In this paper, we extend this line of inquiry by considering oracles S that have *positive dimension* (a complexity-theoretic analog of classical Hausdorff dimension [11,8]) with respect to various resource bounds. Specifically, we prove that every oracle S that has positive Δ_3^{p}-dimension (hence every oracle S that has positive pspace-dimension) satisfies $\mathrm{BPP} \subseteq \mathrm{P}^S$.

Our main theorem is a generalization of this fact that applies to randomized promise classes at various levels of the polynomial-time hierarchy. (Promise problems were introduced by Grollman and Selman [10]. The randomized promise class Promise-BPP was introduced by Buhrman and Fortnow [6] and shown by Fortnow [9] to characterize a "strength level" of derandomization hypotheses. The randomized promise class Promise-AM was introduced by Moser [24].) For every integer $k \geq 0$, our main theorem says that, for every oracle S with positive $\Delta_{k+3}^{\mathrm{p}}$-dimension, every $\mathrm{BP} \cdot \Sigma_k^{\mathrm{P}}$ promise problem is $\Sigma_k^{\mathrm{P},S}$-separable. In particular, if S has positive Δ_3^{p}-dimension, then every BPP promise problem is P^S-separable, and, if S has positive Δ_4^{p}-dimension, then every AM promise problem is NP^S-separable.

We use our results to investigate classes of the form

$$\text{dimalmost-}\mathcal{C} = \left\{ A \;\middle|\; \dim_{\mathrm{H}}(\left\{ B \;\middle|\; A \notin \mathcal{C}^B \right\}) = 0 \right\}$$

for various complexity classes $\mathcal{C}$. It is clear that dimalmost-$\mathcal{C}$ is contained in the extensively investigated class

$$\text{almost-}\mathcal{C} = \left\{ A \;\middle|\; \Pr[A \notin \mathcal{C}^B] = 0 \right\},$$

where the probability is computed according to the uniform distribution (Lebesgue measure) on the set of all oracles B. We show that

$$\text{dimalmost-}\Sigma_k^{\mathrm{P}}\text{-Sep} = \text{almost-}\Sigma_k^{\mathrm{P}}\text{-Sep} = \text{Promise-BP} \cdot \Sigma_k^{\mathrm{P}}$$

holds for every integer $k \geq 0$, where Σ_k^{P}-Sep is the set of all Σ_k^{P}-separable pairs of languages. This implies that

$$\text{dimalmost-P} = \text{BPP},$$

refining the proof by Bennett and Gill [5] that almost-P = BPP. Also, for all $k \geq 1$,

$$\text{dimalmost-}\Sigma_k^{\mathrm{P}} = \mathrm{BP} \cdot \Sigma_k^{\mathrm{P}},$$

refining the proof by Nisan and Wigderson [25] that almost-$\Sigma_k^{\mathrm{P}} = \mathrm{BP} \cdot \Sigma_k^{\mathrm{P}}$.

The 1997 derandomization method of Impagliazzo and Wigderson [16] is central to our arguments.

2 Resource-Bounded Dimension and Relativized Circuit Complexity

This section reviews and develops those aspects of resource-bounded dimension and its relationship to relativized circuit-size complexity that are needed in this paper. It is convenient to use entropy rates as an intermediate step in this development.

2.1 Resource-Bounded Dimension

Resource-bounded dimension is an extension of classical Hausdorff dimension that imposes dimension structure on various complexity classes. There are now several equivalent ways to formulate resource-bounded dimension. Here we sketch the elements of the original formulation that are useful in this paper.

We work in the Cantor-space $\mathbf{C}$ of all infinite binary sequences.

Definition. ([19]) Let $s \in [0, \infty)$.

1. An *s-gale* is a function $d : \{0,1\}^* \to [0,\infty)$ satisfying
$$d(w) = 2^{-s}[d(w0) + d(w1)]$$
for all $w \in \{0,1\}^*$.
2. An s-gale *succeeds* on a sequence $S \in \mathbf{C}$ if $\limsup_{n\to\infty} d(S[0..n-1]) = \infty$, where $S[0..n-1]$ denotes the n-bit prefix of S.
3. The *success set* of an s-gale d is $S^\infty[d] = \{S \in \mathbf{C} \mid d \text{ succeeds on } S\}$.

The following *gale characterization of Hausdorff dimension* is the key to resource-bounded dimension. In this paper we will use this characterization *in place* of the original definition of Hausdorff dimension [11,8], which we refrain from repeating here.

Theorem 2.1 *(Lutz [19]). The Hausdorff dimension of a set $X \subseteq \mathbf{C}$ is*
$$\dim_{\mathrm{H}}(X) = \inf\{s \mid \textit{there is an } s\textit{-gale } d \textit{ such that } X \subseteq S^\infty[d]\}.$$

To extend Hausdorff dimension to complexity classes, we define a *resource bound* to be one of the following classes of functions.

$\mathrm{all} = \{f \mid f : \{0,1\}^* \to \{0,1\}^*\}$
$\mathrm{p} = \left\{f \in \mathrm{all} \;\middle|\; f \text{ is computable in } n^{O(1)} \text{ time}\right\}$
$\Delta_k^{\mathrm{p}} = \mathrm{p}^{\Sigma_{k-1}^{\mathrm{P}}}$ for $k \geq 2$
$\mathrm{pspace} = \left\{f \in \mathrm{all} \;\middle|\; f \text{ is computable in } n^{O(1)} \text{ space}\right\}$

Each of these resource bounds Δ is associated with a *result class* $R(\Delta)$ defined as follows.

$R(\mathrm{all}) = \mathbf{C}$
$R(\mathrm{p}) = \mathrm{E} = \mathrm{TIME}(2^{\mathrm{linear}})$
$R(\Delta_k^{\mathrm{p}}) = \Delta_k^{\mathrm{E}} = \mathrm{E}^{\Sigma_{k-1}^{\mathrm{P}}}$
$R(\mathrm{pspace}) = \mathrm{ESPACE} = \mathrm{SPACE}(2^{\mathrm{linear}})$

A real-valued function $f : \{0,1\}^* \to [0,\infty)$ is *Δ-computable* if there is a function $\hat{f} : \{0,1\}^* \times \mathbb{N} \to \mathbb{Q}$ such that $\hat{f} \in \Delta$ (where the input $(w,r) \in \{0,1\}^* \times \mathbb{N}$ is suitably encoded with r in unary) and, for all $w \in \{0,1\}^*$ and $r \in \mathbb{N}$, $|\hat{f}(w,r) - f(w)| \leq 2^{-r}$.

We now define resource-bounded dimension by imposing resource bounds on the gale characterization in Theorem 2.1.

Definition. ([19]) Let Δ be a resource bound, and let $X \subseteq \mathbf{C}$. (We identify each $S \in X$ with the language whose characteristic sequence is S.)

1. The Δ-*dimension* of X is

$$\dim_\Delta(X) = \inf\{s \mid \text{there is a } \Delta\text{-computable } s\text{-gale } d \text{ such that } X \subseteq S^\infty[d]\}.$$

2. The *dimension* of X *in* $R(\Delta)$ is $\dim(X|R(\Delta)) = \dim_\Delta(X \cap R(\Delta))$.

As shown in [19], these definitions endow the above-mentioned complexity classes $R(\Delta)$ with dimension structure. In general,

$$0 \leq \dim(X|R(\Delta)) \leq \dim_\Delta(X) \leq 1,$$

and $\dim(R(\Delta)|R(\Delta)) = 1$. Also,

$$\Delta \subseteq \Delta' \implies \dim_{\Delta'}(X) \leq \dim_\Delta(X),$$

e.g., $\dim_{\text{pspace}}(X) \leq \dim_{\Delta_3^{\text{P}}}(X)$. It is clear that $\dim_{\text{all}}(X) = \dim(X|\mathbf{C}) = \dim_{\text{H}}(X)$.

Our main results involve Δ-dimensions of individual sequences S, by which we mean

$$\dim_\Delta(S) = \dim_\Delta(\{S\}).$$

We use the easily verified fact that, if Δ is any of the *countable* resource bounds above, then

$$\dim_{\text{H}}(\{S \mid \dim_\Delta(S) = 0\}) = 0. \tag{2.1}$$

For more discussion, motivation, examples, and results, see [19,14,20,12,22].

2.2 Entropy Rates

We use a recent result of Hitchcock and Vinodchandran [15] relating entropy rates to dimension. Entropy rates were studied by Chomsky and Miller [7], Kuich [17], Staiger [26,27], Hitchcock [12], and others.

Definition. The *entropy rate* of a language $A \subseteq \{0,1\}^*$ is

$$H_A = \limsup_{n \to \infty} \frac{\log |A_{=n}|}{n},$$

where $A_{=n} = A \cap \{0,1\}^n$.

Definition. Let $\mathcal{C}$ be a class of languages, and let $X \subseteq \mathbf{C}$. The $\mathcal{C}$-*entropy rate* of X is

$$\mathcal{H}_{\mathcal{C}}(X) = \inf\left\{H_A \;\middle|\; A \in \mathcal{C} \text{ and } X \subseteq A^{\text{i.o.}}\right\},$$

where

$$A^{\text{i.o.}} = \{S \in \mathbf{C} \mid (\exists^\infty n) S[0..n-1] \in A\}.$$

The following result is a routine relativization of Theorem 5.5 of [15].

Theorem 2.2 *(Hitchcock and Vinodchandran [15]). For all $X \subseteq \mathbf{C}$ and $k \in \mathbb{Z}^+$,*

$$\dim_{\Delta_{k+2}^{\text{P}}}(X) \leq \mathcal{H}_{\Sigma_k^{\text{P}}}(X).$$

2.3 Relativized Circuit-Size Complexity

Definition. 1. ([28]). For $f : \{0,1\}^n \to \{0,1\}$ and $A \subseteq \{0,1\}^*$, $\mathrm{size}^A(f)$ is the minimum size of (i.e., number of wires in) an n-input oracle circuit γ such that γ^A computes f.
2. For all $x \in \{0,1\}^*$ and $A \subseteq \{0,1\}^*$, $\mathrm{size}^A(x) = \mathrm{size}^A(f_x)$, where $f_x : \{0,1\}^{\lceil \log |x| \rceil} \to \{0,1\}$ is defined by

$$f_x(w_i) = \begin{cases} x[i] & \text{if } 0 \le i < |x| \\ 0 & \text{if } i \ge |x|, \end{cases}$$

$w_0, \ldots, w_{2^{\lceil \log |x| \rceil}-1}$ lexicographically enumerate $\{0,1\}^{\lceil \log |x| \rceil}$, and $x[i]$ is the ith bit of x.

Lemma 2.3 *For all $A, S \in \mathbf{C}$,*

$$\mathcal{H}_{\mathrm{NP}^A}(\{S\}) \le \liminf_{n\to\infty} \frac{\mathrm{size}^A(S[0..n-1])\log n}{n}.$$

Notation. For $k \in \mathbb{N}$ and $x \in \{0,1\}^*$, we write

$$\mathrm{size}^{\Sigma_k^{\mathrm{P}}}(x) = \mathrm{size}^{K^k}(x),$$

where K^k is the canonical Σ_k^{P}-complete language [4].

By Theorem 2.2 and Lemma 2.3, we have the following.

Theorem 2.4 *For all $S \in \mathbf{C}$ and $k \in \mathbb{N}$,*

$$\dim_{\Delta_{k+3}^{\mathrm{P}}}(S) \le \liminf_{n\to\infty} \frac{\mathrm{size}^{\Sigma_k^{\mathrm{P}}}(S[0..n-1])\log n}{n}$$

3 Positive-Dimension Derandomization

In order to state our main theorem, we review the notion of separability and give a formulation of Promise-BP-classes that is suitable for our purposes.

Definition. Given a class $\mathcal{C}$ of languages, an ordered pair $A = (A^+, A^-)$ of (disjoint) languages is *$\mathcal{C}$-separable* if there exists a language $C \in \mathcal{C}$ such that $A^+ \subseteq C$ and $A^- \cap C = \varnothing$. We write

$$\mathcal{C}\text{-Sep} = \left\{(A^+, A^-) \;\middle|\; (A^+, A^-) \text{ is } \mathcal{C}\text{-separable}\right\}.$$

Definition. Fix a standard paring function $\langle , \rangle : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$.

1. A *witness configuration* is an ordered pair $\mathcal{B} = (B, g)$ where $B \subseteq \{0,1\}^*$ and $g : \mathbb{N} \to \mathbb{N}$.

2. Given a witness configuration $\mathcal{B} = (B, g)$, the *$\mathcal{B}$-critical event* for a string $x \in \{0,1\}^*$ is the set

$$\mathcal{B}_x = \left\{ w \in \{0,1\}^{g(|x|)} \mid \langle x, w \rangle \in B \right\},$$

interpreted as an event in the sample space $\{0,1\}^{g(|x|)}$ with the uniform probability measure. (That is, the probability of $\mathcal{B}_x$ is $\Pr(\mathcal{B}_x) = 2^{-g(|x|)}|\mathcal{B}_x|$.)
3. Given a class $\mathcal{C}$ of languages, we define the class Promise-BP $\cdot\, \mathcal{C}$ to be the set of all ordered pairs $A = (A^+, A^-)$ of languages for which there is a witness configuration $\mathcal{B} = (B, q)$ with the following four properties.
   (i) $B \in \mathcal{C}$.
   (ii) q is a polynomial.
   (iii) For all $x \in A^+$, $\Pr(\mathcal{B}_x) \geq \frac{2}{3}$.
   (iv) For all $x \in A^-$, $\Pr(\mathcal{B}_x) \leq \frac{1}{3}$.

Note that Promise-BP is an operator that maps a class $\mathcal{C}$ of languages to a class Promise-BP $\cdot\, \mathcal{C}$ of disjoint pairs of languages. In particular,

$$\text{Promise-BP} \cdot \text{P} = \text{Promise-BPP}$$

is the class of *BPP promise problems* investigated by Buhrman and Fortnow [6] and Moser [23], and

$$\text{Promise-BP} \cdot \text{NP} = \text{Promise-AM}$$

is the class of *Arthur-Merlin promise problems* investigated by Moser [24].

The following result is the main theorem of this paper.

Theorem 3.1 *For every $S \in \mathbf{C}$ and $k \in \mathbb{N}$,*

$$\dim_{\Delta^{\text{P}}_{k+3}}(S) > 0 \implies \text{Promise-BP} \cdot \Sigma^{\text{P}}_k \subseteq \Sigma^{\text{P},S}_k\text{-Sep}.$$

Theorem 3.1 has many consequences. First, the cases $k = 0$ and $k = 1$ are of particular interest:

Corollary 3.2 *For every $S \in \mathbf{C}$,*

$$\dim_{\Delta^{\text{P}}_3}(S) > 0 \implies \text{Promise-BPP} \subseteq \text{P}^S\text{-Sep}$$

and

$$\dim_{\Delta^{\text{P}}_4}(S) > 0 \implies \text{Promise-AM} \subseteq \text{NP}^S\text{-Sep}.$$

We next note that our results for promise problems imply the corresponding results for decision problems. (Note, however, that the results of Fortnow [9] suggest that the results on promise problems are in some sense stronger.)

Corollary 3.3 *For every $S \in \mathbf{C}$ and $k \in \mathbb{N}$,*

$$\dim_{\Delta^{\text{P}}_{k+3}}(S) > 0 \implies \text{BP} \cdot \Sigma^{\text{P}}_k \subseteq \Sigma^{\text{P},S}_k.$$

In particular,

$$\dim_{\Delta_3^{\mathrm{p}}}(S) > 0 \implies \mathrm{BPP} \subseteq \mathrm{P}^S \tag{3.1}$$

and

$$\dim_{\Delta_4^{\mathrm{p}}}(S) > 0 \implies \mathrm{AM} \subseteq \mathrm{NP}^S. \tag{3.2}$$

Intuitively, (3.1) says that even an oracle S with Δ_3^{p}-dimension 0.001 – which need not be random relative to any reasonable distribution – "contains enough randomness" to carry out a deterministic simulation of BPP. To put the matter differently, to prove that P = BPP, we need "only" show how to dispense with such an oracle S.

As in section 1, for each relativizable complexity class $\mathcal{C}$ (of languages or pairs of languages), define the *dimension-almost-class*

$$\text{dimalmost-}\mathcal{C} = \left\{ A \;\middle|\; \dim_{\mathrm{H}}(\left\{ S \;\middle|\; A \notin \mathcal{C}^S \right\}) = 0 \right\},$$

noting that this is contained in the previously studied *almost-class*

$$\text{almost-}\mathcal{C} = \left\{ A \;\middle|\; \Pr[A \in \mathcal{C}^S] = 1 \right\},$$

where the probability is computed according to the uniform distribution (Lebesgue measure) on the set of all oracles S.

Theorem 3.4 *For every $k \in \mathbb{N}$,*

$$\text{dimalmost-}\Sigma_k^{\mathrm{P}}\text{-Sep} = \text{almost-}\Sigma_k^{\mathrm{P}}\text{-Sep} = \text{Promise-BP} \cdot \Sigma_k^{\mathrm{P}}.$$

Corollary 3.5 *For every $k \in \mathbb{N}$,*

$$\text{dimalmost-}\Sigma_k^{\mathrm{P}} = \mathrm{BP} \cdot \Sigma_k^{\mathrm{P}}.$$

In particular,

$$\text{dimalmost-P} = \mathrm{BPP} \tag{3.3}$$

and

$$\text{dimalmost-NP} = \mathrm{AM}. \tag{3.4}$$

It should be noted that derandomization plays a significantly larger role in the proof of Corollary 3.5 than in the proofs of the analogous results for almost-classes. For example, the proof by Bennett and Gill [5] that almost-P = BPP uses the easily proven fact that the set $X = \left\{ S \;\middle|\; \mathrm{P}^S \neq \mathrm{BPP}^S \right\}$ has Lebesgue measure 0. Hitchcock [13] has recently proven that this set has Hausdorff dimension 1, so the Bennett-Gill argument does not extend to a proof of (3.3). Instead, our proof of (3.3) relies, via (3.1), on Impagliazzo and Wigderson's pseudorandom generator to prove that the set $Y = \left\{ S \;\middle|\; \mathrm{BPP} \not\subseteq \mathrm{P}^S \right\}$ has Hausdorff dimension 0. Similarly, the proof by Nisan and Wigderson [25] that almost-NP $\subseteq$ AM uses derandomization, but their proof that AM $\subseteq$ almost-NP is elementary. In contrast, *both* directions of the proof of (3.4) use derandomization: The inclusion dimalmost-NP $\subseteq$ AM relies on the fact that almost-NP $\subseteq$ AM (hence on derandomization), and our proof that AM $\subseteq$ dimalmost-NP relies, via (3.2), on Impagliazzo and Wigderson's pseudorandom generator.

4 Conclusion

We conclude with a brief remark on relativization. Impagliazzo and Wigderson's pseudorandom generator exists relative to arbitrary oracles, as to all our arguments here. For example, implication (3.1),

$$\dim_{\Delta_3^{\mathrm{p}}}(S) > 0 \implies \mathrm{BPP} \subseteq \mathrm{P}^S,$$

holds relative to every oracle. Note, however, that, if we consider this implication relative to an oracle S of positive Δ_3^{p}-dimension, then the *relativized* Δ_3^{p}-dimension of this S will be 0, so we cannot use the relativized implication to conclude that $\mathrm{P}^S = \mathrm{BPP}^S$. Indeed, by Hitchcock's just-mentioned result and (2.1), there must exist languages S of positive Δ_3^{p}-dimension for which $\mathrm{P}^S \neq \mathrm{BPP}^S$.

Acknowledgments. We are grateful to Eric Allender, whose discussions of the ideas in [1,2] were a significant motivation of our work. We thank an anonymous referee for pointing out a flaw in our earlier proof. We thank John Hitchcock for useful discussions and suggestions. We also thank Manindra Agrawal for a useful discussion.

References

1. E. Allender. When worlds collide: Derandomization, lower bounds, and kolmogorov complexity. In *Proceedings of the 21st annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2245 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2001.
2. E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 669–678, 2002. *SIAM Journal on Computing.* To appear.
3. E. Allender and M. Strauss. Measure on small complexity classes with applications for BPP. In *Proceedings of the 35th Symposium on Foundations of Computer Science*, pages 807–818, 1994.
4. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, Berlin, second edition, 1995.
5. C. H. Bennett and J. Gill. Relative to a random oracle A, $\mathrm{P}^A \neq \mathrm{NP}^A \neq \text{co-NP}^A$ with probability 1. *SIAM Journal on Computing*, 10:96–113, 1981.
6. H. Buhrman and L. Fortnow. One-sided versus two-sided randomness. In *Proceedings of the sixteenth Symposium on Theoretical Aspects of Computer Science*, pages 100–109, 1999.
7. N. Chomsky and G. A. Miller. Finite state languages. *Information and Control*, 1(2):91–112, 1958.
8. K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications.* Wiley, second edition, 2003.
9. L. Fortnow. Comparing notions of full derandomization. In *Proceedings of the 16th IEEE Conference on Computational Complexity*, pages 28–34, 2001.
10. J. Grollman and A. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 11:309–335, 1988.

11. F. Hausdorff. Dimension und äusseres Mass. *Mathematische Annalen*, 79:157–179, 1919.
12. J. M. Hitchcock. *Effective fractal dimension: foundations and applications*. PhD thesis, Iowa State University, 2003.
13. J. M. Hitchcock. Hausdorff dimension and oracle constructions. *Theoretical Computer Science*, 355(3):382–388, 2006.
14. J. M. Hitchcock, J. H. Lutz, and E. Mayordomo. The fractal geometry of complexity classes. *SIGACT News*, 36(3):24–38, 2005.
15. J. M. Hitchcock and N. V. Vinodchandran. Dimension, entropy rates, and compression. *Journal of Computer and System Sciences*, 72(4):760–782, 2006.
16. R. Impagliazzo and A. Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th Symposium on Theory of Computing*, pages 220–229, 1997.
17. W. Kuich. On the entropy of context-free languages. *Information and Control*, 16(2):173–200, 1970.
18. J. H. Lutz. A pseudorandom oracle characterization of BPP. *SIAM Journal on Computing*, 22(5):1075–1086, 1993.
19. J. H. Lutz. Dimension in complexity classes. *SIAM Journal on Computing*, 32:1236–1259, 2003.
20. J. H. Lutz. Effective fractal dimensions. *Mathematical Logic Quarterly*, 51:62–72, 2005.
21. P. Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.
22. E. Mayordomo. Effective Hausdorff dimension. In *Proceedings of Foundations of the Formal Sciences III*, pages 171–186. Kluwer Academic Press, 2004.
23. P. Moser. Relative to P promise-BPP equals APP. Technical Report TR01-68, Electronic Colloquium on Computational Complexity, 2001.
24. P. Moser. Random nondeterministic real functions and Arthur Merlin games. Technical Report TR02-006, ECCC, 2002.
25. N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.
26. L. Staiger. Kolmogorov complexity and Hausdorff dimension. *Information and Computation*, 103:159–94, 1993.
27. L. Staiger. A tight upper bound on Kolmogorov complexity and uniformly optimal prediction. *Theory of Computing Systems*, 31:215–29, 1998.
28. C. B. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, 31:169–181, 1985.

Approximation Algorithms and Hardness Results for Labeled Connectivity Problems

Refael Hassin[1], Jérôme Monnot[2], and Danny Segev[1]

[1] School of Mathematical Sciences, Tel-Aviv University, Israel
{hassin, segevd}@post.tau.ac.il
[2] CNRS LAMSADE, Université Paris-Dauphine, France
monnot@lamsade.dauphine.fr

Abstract. Let $G = (V, E)$ be a connected multigraph, whose edges are associated with labels specified by an integer-valued function $\mathcal{L} : E \rightarrow \mathbb{N}$. In addition, each label $\ell \in \mathbb{N}$ to which at least one edge is mapped has a non-negative cost $c(\ell)$. The *minimum label spanning tree* problem (MinLST) asks to find a spanning tree in G that minimizes the overall cost of the labels used by its edges. Equivalently, we aim at finding a minimum cost subset of labels $I \subseteq \mathbb{N}$ such that the edge set $\{e \in E : \mathcal{L}(e) \in I\}$ forms a connected subgraph spanning all vertices. Similarly, in the *minimum label s-t path* problem (MinLP) the goal is to identify an s-t path minimizing the combined cost of its labels, where s and t are provided as part of the input.

The main contributions of this paper are improved approximation algorithms and hardness results for MinLST and MinLP. As a secondary objective, we make a concentrated effort to relate the algorithmic methods utilized in approximating these problems to a number of well-known techniques, originally studied in the context of integer covering.

1 Introduction

The majority of graph connectivity problems have traditionally been studied under the assumption that each edge is associated with a *numerical attribute*, to which we refer as length, weight or cost, depending on the related real-life context. In this long-established model, the computational task is to identify a subgraph satisfying given connectivity requirements, with the objective of minimizing some function defined over the lengths of picked edges. While these settings capture a wide range of practical scenarios, they nevertheless fail to incorporate *grouping constraints* stating that the set of available edges is partitioned into classes, each of which can be purchased in its entirety or not at all. A rather convenient way of integrating grouping constraints is to couple each edge with a *label* that specifies its class. Having this extra notation at hand, we say that a subset of labels forms a feasible solution when the edges whose labels belong to this subset induce a subgraph satisfying the given connectivity requirements. Since costs are now assigned to labels rather than to single edges, the objective is to find a solution that minimizes some function defined over the costs of picked labels.

We address two of the most fundamental labeled connectivity problems, those of constructing spanning trees and s-t paths by picking labels of minimum total cost. Formally, let $G = (V, E)$ be a connected multigraph on n vertices, whose edges are associated with labels specified by an integer-valued function $\mathcal{L} : E \rightarrow \mathbb{N}$. In addition, each label $\ell \in \mathbb{N}$ to which at least one edge is mapped has a non-negative cost $c(\ell)$. The *minimum label spanning tree* problem (MinLST) asks to find a spanning tree in G that minimizes the overall cost of the labels used by its edges. Equivalently, we aim at finding a minimum cost subset of labels $I \subseteq \mathbb{N}$ such that the edge set $\{e \in E : \mathcal{L}(e) \in I\}$ forms a connected subgraph spanning all vertices. Similarly, in the *minimum label s-t path* problem (MinLP) the goal is to identify an s-t path minimizing the combined cost of its labels, where s and t are provided as part of the input. We refer to the special cases of these problems in which at most r edges are assigned to any given label as MinLST$_r$ and MinLP$_r$, respectively.

1.1 Related Results

Prior to describing the line of work preceding the current paper, we remark that, to the best of our knowledge, the weighted version of both MinLST and MinLP has not been previously studied. Therefore, the reader should bear in mind that the undermentioned upper and lower bounds on the approximability of these problems are stated with respect to the unweighted case, in which each label has a unit cost.

Chang and Leu [9] seem to have been the first to consider MinLST. They proved that the corresponding decision problem is NP-complete, and experimentally studied the performance of several heuristics, one of which is the *maximum vertex covering* algorithm. Krumke and Wirth [16] demonstrated that a variant of this algorithm (henceforth, modified MVC) guarantees an approximation factor of at most $2 \ln n + 1$, and accompanied this result by a hardness proof showing that MinLST is at least as hard to approximate as *set cover*. Wan, Chen and Xu [21] suggested a refined analysis of the modified MVC algorithm to obtain a factor of at most $\mathcal{H}_{n-1}$. Very recently, Xiong, Golden and Wasil [23] established that this algorithm provides a tight approximation guarantee of $\mathcal{H}_r$ for MinLST$_r$, improving the bound of Wan et al.[1], which is independent of r. Brüggemann, Monnot and Woeginger [7] considered a local-search heuristic, and showed that it constructs a solution for MinLST$_r$ whose cost is within factor $\frac{r+1}{2}$ of optimum. In addition, they proved that MinLST$_2$ is polynomial-time solvable, whereas MinLST$_r$ is APX-complete for $r \geq 3$.

Carr, Doddi, Konjevod and Marathe [8] proved that MinLP contains as a special case the *red-blue set cover* problem, which was shown in the same paper to be inapproximable within a factor of $O(2^{\log^{1-\epsilon} n})$ for any $\epsilon > 0$, unless NP $\subseteq$ TIME$(n^{\text{polylog}(n)})$. However, this hardness result does not readily extend to MinLP, since the reduction described by Carr et al. is not approximation preserving. Relying on a more restrictive subproblem of red-blue set cover, Wirth [22, Thm. 2.16] established the above-mentioned lower bound for MinLP. On the positive side, Broersma, Li, Woeginger and Zhang [6] devised two exact exponential-time algorithms, with respective running times of $O(n \cdot \min\{L^d, 2^L\})$ and $O(n^2 L!)$, where L is the number of labels and d is the s-t distance in G. They also considered a Dijkstra-like algorithm for approximating MinLP, and demonstrated that it does not provide any constant factor. In fact, simple examples show that the resulting solution may have a cost of $\Omega(n)$ times the optimum, and moreover, to our knowledge a non-trivial approximation for MinLP has not been presented yet.

[1] Note that we may assume that $r \leq n - 1$, since in the opposite case MinLST$_r$ can be reduced to MinLST$_{n-1}$ by eliminating an edge from each uniform labeled cycle.

1.2 Our Results

In this paper, we present improved approximation algorithms and hardness results for MinLST and MinLP. As a secondary objective, we make a concentrated effort to relate the algorithmic methods utilized in approximating these problems to a number of well-known techniques, originally studied in the context of integer covering. Our main findings can be briefly summarized as follows:

1. We extend the modified MVC algorithm to handle label costs in MinLST. Consequently, we derive the first algorithm for the weighted case, and prove that its approximation guarantee is $\mathcal{H}_{n-1}$. This result appears in Section 2.
2. We provide an additional $O(\log n)$ approximation for MinLST, which is based on assembling partial solutions obtained by repeatedly calling a constant-factor *maximum coverage* subroutine [1,15,20]. This approach encapsulates the principal idea we employ to approximate MinLP, and its specifics are described in Section 2.
3. By prematurely terminating the modified MVC heuristic and switching to an exact algorithm for MinLST$_2$ (due to Brüggemann et al. [7]), we achieve an approximation factor of $\mathcal{H}_r - \frac{1}{6}$ for unweighted MinLST$_r$. Our algorithm was inspired by a similar improvement for the set cover problem, proposed by Goldschmidt, Hochbaum and Yu [11]. In addition to showing that the factor $\mathcal{H}_r$ can be decreased by lower order terms, the underlying analysis we present is considerably simpler than that of Xiong et al. [23]. This algorithm is given in the full version of this paper [13].
4. We devise the first non-trivial algorithm for MinLP, with an approximation factor of $O(\sqrt{n})$. A crucial ingredient of this algorithm is a preprocessing step, in which we "guess" certain attributes of an arbitrary optimal solution and modify the given instance accordingly. Once again, we make use of repeated calls to a maximum coverage subroutine, eventually allowing us to easily identify a near-optimal solution. This result is described in Section 3.
5. Since MinLP$_r$ admits a constant-factor approximation when $r = O(1)$, one may ask whether MinLP$_r$ can be approximated in this case to any required degree. A negative answer to this question is provided in Section 4. Specifically, we show that MinLP$_r$ is at least as hard to approximate as Min-r-SAT, a special case of the *minimum satisfiability* problem (MinSAT) in which each clause consists of at most r literals. The inapproximability of the former problem was studied by Avidor and Zwick [4], whereas that of MinSAT was studied even earlier by Marathe and Ravi [18].
6. By utilizing a self-improvability property of MinLP, which is based on the notion of *label squaring*, we show that MinLP cannot be approximated to within any polylogarithmic factor unless P = NP. This result is incomparable with the previously mentioned lower bound of Wirth, stating that an approximation factor of $O(2^{\log^{1-\epsilon} n})$ for some $\epsilon > 0$ implies NP $\subseteq$ TIME($n^{\text{polylog}(n)}$). Our technique was motivated by an analogous construction due to Karger, Motwani and Ramkumar [14] for the longest path problem. This proof is given in Section 4.

Due to space limitations, several proofs are omitted from this extended abstract. We refer the reader to the full version of this paper [13], in which all missing proofs are provided.

1.3 Notation

We conclude this section by introducing some notation and terminology. Given a set of edges $F \subseteq E$, we use $\mathcal{L}(F) = \{\mathcal{L}(e) : e \in F\}$ to denote the image of F under $\mathcal{L}$. Furthermore, when H is a subgraph of G, the notation $\mathcal{L}(H)$ is used as a shorthand for $\mathcal{L}(E(H))$. For a subset $I \subseteq \mathbb{N}$, we denote by $\mathcal{L}^{-1}(I) = \{e \in E : \mathcal{L}(e) \in I\}$ the inverse image of I, excluding the case where the specified subset is actually a singleton, which is abbreviated by writing $\mathcal{L}^{-1}(\ell)$ instead of $\mathcal{L}^{-1}(\{\ell\})$. The *contraction* of an edge (u, v) is the multigraph obtained by identifying the vertices u and v, followed by eliminating any degenerate edge joining the newly created vertex to itself. It is easy to verify that, regardless of the order according to which the edges in a subset $F \subseteq E$ are contracted, we always attain the same multigraph. Therefore, it is sensible to define the contraction of an edge set.

2 Approximating the Weighted MinLST Problem

In what follows, we present two approximation algorithms for the MinLST problem in its utmost generality, where each label has a non-negative cost. Guided by considerably different techniques, both algorithms iteratively construct a feasible subset of labels whose cost is within a factor of $O(\log n)$ of optimum. We remind the reader that MinLST is at least as hard to approximate as set cover, implying that the factor we derive is best possible up to a constant multiplicative factor, assuming P $\neq$ NP [3,19].

2.1 The Greedy Algorithm

We extend the modified MVC algorithm, originally suggested by Krumke and Wirth [16], to handle label costs. In each step, our algorithm picks the most cost-effective label, namely, one that minimizes the ratio between its cost and the decrement in the number of vertices resulting from the contraction of its corresponding edges. A formal description of the algorithm is provided in Figure 1, followed by a tight analysis showing that its approximation guarantee is exactly $\mathcal{H}_{n-1}$.

Theorem 1. *The cost of the constructed solution is within a factor of $\mathcal{H}_{n-1}$ of optimum.*

Proof. Let $\{\ell_1, \ldots, \ell_k\}$ be the set of labels returned by the algorithm, indexed by the order in which they were picked. In addition, for $1 \leq j \leq k$, let H_j be the processed multigraph at the beginning of the jth iteration (in which the label ℓ_j was picked). In what follows, we denote by OPT the cost of an optimal solution to the original instance, and by OPT(H_j) the cost of an optimal solution to the instance we obtain at the beginning of the jth iteration. Clearly, OPT = OPT$(H_1) \geq \cdots \geq$ OPT(H_k).

We first show that $c(\ell_j) \leq d_{H_j}(\ell_j)\frac{\text{OPT}(H_j)}{|V(H_j)|-1}$ for all $1 \leq j \leq k$. Let $\{\ell_1^*, \ldots, \ell_p^*\} \subseteq \mathcal{L}(H_j)$ be an optimal solution to the instance corresponding to OPT(H_j). Note that the algorithm had the option of picking each ℓ_i^* when ℓ_j was picked. By observing that a minimum-ratio label is picked in each iteration, we have $\frac{c(\ell_j)}{d_{H_j}(\ell_j)} \leq \frac{c(\ell_i^*)}{d_{H_j}(\ell_i^*)}$ for every $1 \leq i \leq p$, and the stated upper bound on $c(\ell_j)$ follows as

$$\text{OPT}(H_j) = \sum_{i=1}^{p} c(\ell_i^*) \geq \frac{c(\ell_j)}{d_{H_j}(\ell_j)} \sum_{i=1}^{p} d_{H_j}(\ell_i^*) \geq \frac{c(\ell_j)}{d_{H_j}(\ell_j)} \left(|V(H_j)| - 1\right) .$$

```
1. $I \leftarrow \emptyset$, $H \leftarrow G$.
2. While $H$ contains at least two vertices
   (a) For every label $\ell \in \mathcal{L}(H)$, let $d_H(\ell)$ be the decrement in the number of vertices in $H$
       when the edge set $\mathcal{L}^{-1}(\ell)$ is contracted.
   (b) Pick a label $\ell^*$ that minimizes the ratio $\frac{c(\ell)}{d_H(\ell)}$ over all labels in $\mathcal{L}(H)$.
   (c) $I \leftarrow I \cup \{\ell^*\}$, $H \leftarrow$ the contraction of $\mathcal{L}^{-1}(\ell^*)$ in $H$.
3. Return $I$.
```

Fig. 1. The greedy algorithm

The second inequality holds since the set of edges $\mathcal{L}^{-1}(\{\ell_1^*, \dots, \ell_p^*\})$ forms a connected subgraph spanning $V(H_j)$, implying that $\sum_{i=1}^{p} d_{H_j}(\ell_i^*) \geq |V(H_j)| - 1$.

Using the upper bounds proved above, we conclude that

$$\sum_{j=1}^{k} c(\ell_j) \leq \sum_{j=1}^{k} d_{H_j}(\ell_j) \frac{\mathrm{OPT}(H_j)}{|V(H_j)| - 1} \leq \mathrm{OPT} \sum_{j=1}^{k} \sum_{i=1}^{d_{H_j}(\ell_j)} \frac{1}{|V(H_j)| - i} = \mathcal{H}_{n-1} \cdot \mathrm{OPT} \ ,$$

where the last equality holds since $d_{H_j}(\ell_j) = |V(H_j)| - |V(H_{j+1})|$. □

Lemma 2. *There are MinLST instances for which the algorithm produces a solution whose cost is $\mathcal{H}_{n-1}$ times the optimum.*

2.2 The Budgeted Covering Algorithm

Unlike the shortsighted approach employed by the greedy algorithm, that picks a single label in each step, the new strategy we suggest consists of repeatedly contracting an inexpensive collection of labels in an attempt to decrease the number of vertices by a constant fraction. Such a collection is identified by approximating a related instance of the budgeted maximum coverage problem, in which we are given a ground set U, a family $\mathcal{S}$ of subsets of U with non-negative costs, and a budget B. The objective is to find a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that the total cost of the subsets in $\mathcal{S}'$ is at most B, and such that the number of elements covered by $\mathcal{S}'$ is maximized. Several algorithms achieve an approximation guarantee of $1 - \frac{1}{e}$ for the latter problem [1,15,20].

To simplify the description and analysis of the budgeted covering algorithm, given in Figure 2, it would be convenient to make two preliminary assumptions. First, we assume that $c_{\min} = \min_{\ell \in \mathcal{L}(G)} c(\ell) > 0$, as all zero cost labels can be picked and contracted in advance. Second, given an accuracy requirement $\epsilon > 0$, we assume that a parameter $\Delta \in [\mathrm{OPT}, (1+\epsilon)\mathrm{OPT}]$ is known. This follows from observing that $c_{\min} \leq \mathrm{OPT} \leq |\mathcal{L}(G)| c_{\max}$, where $c_{\max} = \max_{\ell \in \mathcal{L}(G)} c(\ell)$, so all $O(\log_{1+\epsilon} \frac{|\mathcal{L}(G)| c_{\max}}{c_{\min}})$ candidate values of the form $(1+\epsilon)^k c_{\min}$ can be tested as the correct guess for Δ.

Theorem 3. *The cost of the solution constructed by the budgeted covering algorithm is within a factor of* $(1+\epsilon)\log_{10/7} n$ *of optimum.*

Proof. Starting with an empty set of labels, in each iteration we augment I with labels whose total cost is at most $\Delta \leq (1+\epsilon)\mathrm{OPT}$. Therefore, it is sufficient to show that the

```
1. $I \leftarrow \emptyset$, $H \leftarrow G$.
2. While $H$ contains at least two vertices
   (a) Create a budgeted maximum coverage instance by: The ground set is $V(H)$; for each label $\ell \in \mathcal{L}(H)$ there is a corresponding subset $V_\ell \subseteq V(H)$, consisting of all endpoints of edges in $\mathcal{L}^{-1}(\ell)$; the cost of $V_\ell$ is $c(\ell)$; and the budget is $\Delta$.
   (b) Approximate the instance defined above, to obtain a subset $I' \subseteq \mathcal{L}(H)$.
   (c) $I \leftarrow I \cup I'$, $H \leftarrow$ the contraction of $\mathcal{L}^{-1}(I')$ in $H$.
3. Return $I$.
```

Fig. 2. The budgeted covering algorithm

algorithm terminates within $\log_{10/7} n$ iterations. To this end, we argue that contracting each of the label sets we obtain in step 2b decreases the number of vertices in the processed multigraph by a factor of at least 0.3.

Let $I^* \subseteq \mathcal{L}(G)$ be an optimal solution, with $\sum_{\ell \in I^*} c(\ell) = \text{OPT} \leq \Delta$. Now consider a single iteration. Since $\mathcal{L}^{-1}(I^*)$ forms a connected subgraph of G spanning all vertices, it follows that $\{V_\ell : \ell \in I^* \cap \mathcal{L}(H)\}$ is a feasible solution to the budgeted maximum coverage instance defined in step 2a that fully covers $V(H)$. Consequently, for the current approximate solution I' we must have $|\bigcup_{\ell \in I'} V_\ell| \geq (1 - \frac{1}{e})|V(H)|$, implying that the contraction of $\mathcal{L}^{-1}(I')$ decreases the number of vertices by at least $\frac{1}{2}(1 - \frac{1}{e})|V(H)| > 0.3|V(H)|$. □

3 An $O(\sqrt{n})$ Approximation for MinLP

In what follows, we present the first non-trivial algorithm for the MinLP problem, achieving an approximation factor of $O(\sqrt{n})$. Throughout this section, we assume that the reader is familiar with the basics of budgeted maximum coverage given in Subsection 2.2.

The principal idea that guides our algorithm can be informally described as follows. When s and t are distant enough, an optimal solution must traverse many vertices, a fact that establishes the existence of an inexpensive set of labels whose contraction significantly decreases the number of vertices. As demonstrated in the context of the budgeted covering algorithm, we can identify a label set possessing this property by employing a maximum coverage subroutine. In the opposite case, a shortest path connecting s and t constitutes a near-optimal solution, provided that its edges are not endowed with overly priced labels. These observations suggest a two-step approach: First, perform repeated contractions as long as s and t are distant, and then complete the solution by picking a shortest s-t path.

For this tactic to have a low order strongly-polynomial running time, we apply a technique that was originally proposed by Hassin [12] and enhanced by Lorenz and Raz [17]. In adherence to standard terminology, we define an *α-test* to be a procedure that, given a parameter $\Delta \geq 0$, either constructs a feasible solution whose cost is at most $\alpha\Delta$ or determines that $\text{OPT} > \Delta$. The specifics of a $\frac{13}{3}\sqrt{n}$-test are provided in Figure 3, followed by a correctness proof.

```
1. $I \leftarrow \emptyset$, $H \leftarrow G$.
2. Eliminate from $H$ all edges $e$ with $c(\mathcal{L}(e)) > \Delta$.
3. While $\text{dist}_H(s,t) \geq \sqrt{n}$
   (a) Create a budgeted maximum coverage instance by: The ground set is $V(H)$; for each
       label $\ell \in \mathcal{L}(H)$ there is a corresponding subset $V_\ell \subseteq V(H)$, consisting of all endpoints
       of edges in $\mathcal{L}^{-1}(\ell)$; the cost of $V_\ell$ is $c(\ell)$; and the budget is $\Delta$.
   (b) Approximate the instance defined above, to obtain a subset $I' \subseteq \mathcal{L}(H)$.
   (c) $I \leftarrow I \cup I'$, $H \leftarrow$ the contraction of $\mathcal{L}^{-1}(I')$ in $H$.
4. If the number of iterations in step 3 was greater than $\frac{10}{3}\sqrt{n}$, return "OPT > $\Delta$".
5. Let $P$ be a shortest $s$-$t$ path in $H$. Return $I \cup \mathcal{L}(P)$.
```

Fig. 3. The MinLP test

Lemma 4. *The above procedure is a $\frac{13}{3}\sqrt{n}$-test.*

Now let c_{st} be the minimum label cost for which the subgraph $(V, \{e : c(\mathcal{L}(e)) \leq c_{st}\})$ contains an s-t path. Clearly, $c_{st} \leq \text{OPT} \leq |\mathcal{L}(G)|c_{st}$. Given an accuracy requirement $\epsilon > 0$, we conduct a binary search over $\{(1+\epsilon)^k c_{st} : 0 \leq k \leq \lceil \log_{1+\epsilon} |\mathcal{L}(G)| \rceil\}$, involving $O(\log \log_{1+\epsilon} |\mathcal{L}(G)|)$ calls to the $\frac{13}{3}\sqrt{n}$-test described above. As a consequence, we identify a value Δ for which the test reports OPT $> \Delta$, whereas for $(1+\epsilon)\Delta$ it successfully constructs a feasible solution. It follows that the cost of this solution is at most $(1+\epsilon)\frac{13}{3}\sqrt{n} \cdot \text{OPT}$.

Theorem 5. *For any fixed $\epsilon > 0$, MinLP can be approximated to within a factor of $(1+\epsilon)\frac{13}{3}\sqrt{n}$.*

4 The Hardness of Approximating MinLP

The main result of this section is a hardness proof showing that MinLP cannot be approximated to within any polylogarithmic factor unless P = NP. Prior to presenting this proof, we relate the approximability of MinLP$_r$ to that of the Min-r-SAT problem. It is worth noting that all forthcoming results are proved for the unweighted version of the corresponding problem.

4.1 MinLP$_r$ and Min-r-SAT

The input to the *minimum satisfiability* problem (MinSAT) is a Boolean formula in conjugative normal form, consisting of a collection $C = \{C_1, \ldots, C_m\}$ of clauses made up of complemented and uncomplemented occurrences of variables from the set $X = \{x_1, \ldots, x_n\}$. The objective is to find a truth assignment to the variables that minimizes the number of satisfied clauses. We refer to the special case of this problem, in which each clause has at most r literals, as Min-r-SAT.

Marathe and Ravi [18] showed that MinSAT and vertex cover are equivalent with respect to approximability. Therefore, it is NP-hard to approximate the general MinSAT problem to within any factor smaller than 1.3606 [10]. Having observed that this bound does not apply to Min-r-SAT for small values of r, Avidor and Zwick [4] provided a lower bound of $\frac{15}{14}$ for $r = 2$, and a bound of $\frac{7}{6}$ for all $r \geq 3$. The next theorem extends these results to MinLP_r.

Theorem 6. *For every $r \geq 2$, $MinLP_r$ is at least as hard to approximate as Min-r-SAT.*

Proof. Given an instance (C, X) of Min-r-SAT, we show how to formulate it as a MinLP_r instance. For $1 \leq j \leq n$, let d_j and $\bar{d}_j$ be the number of clauses in which the literals x_j and $\bar{x}_j$ appear, respectively. Without loss of generality, $d_j \geq 1$ and $\bar{d}_j \geq 1$, or otherwise the value of x_j can be determined in advance. We define a MinLP_r instance $(G, \mathcal{L}, s, t)$ as follows:

1. The vertices of G are $v_1, \ldots, v_{n+1}$. In addition, for every $1 \leq j \leq n$, we create two interior-disjoint paths, P_j and $\bar{P}_j$, connecting v_j and v_{j+1}. The length of P_j is d_j, while that of $\bar{P}_j$ is $\bar{d}_j$.
2. We now spread the labels $\{\ell_1, \ldots, \ell_m\}$ on the edges of G. Specifically, let $C(x_j)$ and $C(\bar{x}_j)$ be the sets of clauses in C containing the literals x_j and $\bar{x}_j$, respectively. Then, each edge of P_j is given a distinct label from $\{\ell_i : C_i \in C(x_j)\}$, and similarly, the edges of $\bar{P}_j$ are given distinct labels from $\{\ell_i : C_i \in C(\bar{x}_j)\}$. Since each clause has at most r literals, the number of occurrences of each label is at most r.
3. Finally, we set $s = v_1$ and $t = v_{n+1}$.

We note that there is a one-to-one correspondence between truth assignments and s-t paths in G. First, suppose that f is a truth assignment that satisfies k clauses. Then the concatenation P of the paths $\{P_j : f(x_j) = \mathit{true}\}$ and $\{\bar{P}_j : f(x_j) = \mathit{false}\}$ forms an s-t path with $|\mathcal{L}(P)| = k$. Conversely, suppose that P is an s-t path with $|\mathcal{L}(P)| = k$. Then, as a result of setting each variable x_j to *true* if and only if P_j is a subpath of P, we obtain an assignment satisfying k clauses. □

4.2 Inapproximability Within Any Polylogarithmic Factor

In what follows, we prove that it is NP-hard to approximate MinLP within a factor of $O(\log^k n)$, for any fixed $k \geq 1$. To simplify the presentation, our proof is decomposed into three stages. First, we provide a logarithmic lower bound on the approximability of MinLP by relating it to a subproblem of set cover. Then, we define a new graph operation, called label squaring, and use it to derive a self-improvability property. Finally, we establish the main result by exploiting this property and additional structure common to instances obtained from the reduction described in the first stage.

Lemma 7. *There exists a constant $c > 0$, such that a polynomial time algorithm approximating MinLP within a factor of $c \ln n$ implies* $\text{P} = \text{NP}$.

Proof. By plugging the proof system of Raz and Safra [19] (or alternatively, Arora and Sudan [3]) into the reduction of Bellare, Goldwasser, Lund and Russell [5], the former authors showed that set cover is NP-hard to approximate within a factor of $O(\log n)$. In other words, there is a constant $c' > 0$ such that approximating set cover in polynomial time within a factor of $c' \ln n$ implies $\text{P} = \text{NP}$. This result also applies to instances $(U, \mathcal{S})$ with $|U| > |\mathcal{S}|^{1/q}$, for some constant $q \geq 1$, since the above-mentioned construction guarantees that $|U|$ and $|\mathcal{S}|$ are polynomially related [2]. We refer to this special case as MinSC′.

Given a MinSC′ instance, consisting of a ground set $U = \{e_1, \dots, e_n\}$ and a family of subsets $\mathcal{S} = \{S_1, \dots, S_m\} \subseteq 2^U$, we define an instance $(G, \mathcal{L}, s, t)$ of MinLP as follows:

1. The vertices of G are $v_0, \dots, v_n$. In addition, for each element $e_j \in U$ we create a gadget $G(e_j)$ by connecting v_{j-1} and v_j to the upper rung of a ladder graph. More precisely, if e_j belongs to p subsets in $\mathcal{S}$, we put together a ladder whose rungs are $(a_j^1, b_j^1), \dots, (a_j^p, b_j^p)$, adding the edges (v_{j-1}, a_j^1) and (v_j, b_j^1). This configuration is illustrated in Figure 4.

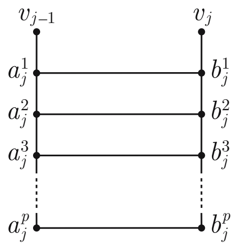


Fig. 4. The gadget $G(e_j)$

2. We now spread the labels $\{\ell_0, \ell_1, \dots, \ell_m\}$ on the edges of G. Using the notation of item 1, each of the p rungs is given a distinct label from $\{\ell_i : e_j \in S_i\}$, whereas all other edges of $G(e_j)$ are given the label ℓ_0.
3. We set $s = v_0$ and $t = v_n$.

At this point, it is imperative to remark that since $n > m^{1/q}$, the above construction ensures that the overall number of vertices satisfies

$$|V(G)| \leq n + 1 + 2nm \leq n + 1 + 2n^{q+1} \leq 4n^{q+1} \ .$$

Now let $c = \frac{c'}{4(q+1)}$, and suppose that MinLP can be approximated in polynomial time within a factor of $c \ln |V(G)|$. We show that this assumption leads to an approximation factor of at most $c' \ln n$ for MinSC′, implying P = NP. To this end, let $\mathcal{S}^* \subseteq \mathcal{S}$ be an optimal solution to the instance $(U, \mathcal{S})$. As all elements of U are covered by $\mathcal{S}^*$, the label of at least one rung in each of the n ladders belongs to $\{\ell_i : S_i \in \mathcal{S}^*\}$, and by augmenting this label set with ℓ_0 we obtain a subgraph of G that contains an s-t path. Therefore, the number of labels in an optimal solution to the new MinLP instance is at most $|\mathcal{S}^*| + 1$. It follows that we can find in polynomial time an s-t path P satisfying

$$\begin{aligned} |\mathcal{L}(P)| &\leq \frac{c'}{4(q+1)} \ln |V(G)| \cdot (|\mathcal{S}^*| + 1) \leq \frac{c'}{4(q+1)} \ln \left(4n^{q+1}\right) \cdot 2|\mathcal{S}^*| \\ &\leq \frac{c'}{2} \ln (4n) \cdot |\mathcal{S}^*| \leq c' \ln n \cdot |\mathcal{S}^*| \ . \end{aligned}$$

The second inequality holds since $|V(G)| \leq 4n^{q+1}$, and the last inequality follows from observing that we can assume without loss of generality that $n \geq 4$, so $\ln(4n) \leq 2 \ln n$. It is not difficult to verify that, as the path P necessarily traverses ℓ_0-labeled edges, $\{S_i \in \mathcal{S} : \ell_i \in \mathcal{L}(P)\}$ is a cover of U with cardinality at most $|\mathcal{L}(P)| - 1 \leq c' \ln n \cdot |\mathcal{S}^*|$. □

Given a MinLP instance $I = (G, \mathcal{L}, s, t)$, its *label squaring* $I^2 = (G^2, \mathcal{L}^2, s^2, t^2)$ is a new instance defined as follows. To assemble the graph G^2, we first construct a distinct copy G_e of G for each original edge $e \in E(G)$. Letting s_e and t_e denote the vertices of G_e that correspond to s and t, we arbitrarily assign s_e and t_e to different endpoints of e. Then, for each $v \in V(G)$, the vertices assigned to v are unified, over all copies, to a single vertex v^2. Using this notation, the source and destination are s^2 and t^2, respectively. Finally, the new set of labels is $\mathcal{L}(G) \times \mathcal{L}(G)$, where the edge of G_e corresponding to $f \in E(G)$ is given the label $(\mathcal{L}(e), \mathcal{L}(f))$.

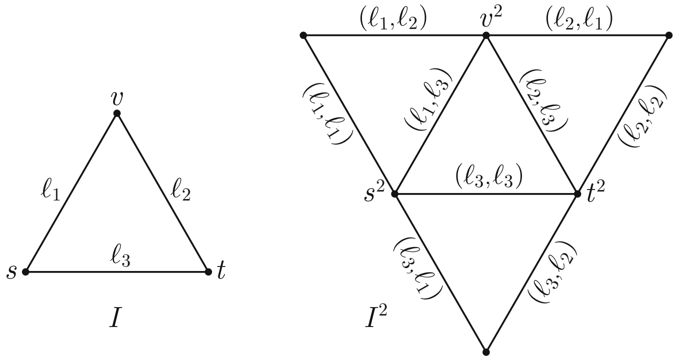


Fig. 5. A label squaring example

Lemma 8. $\mathrm{OPT}(I^2) \le \mathrm{OPT}^2(I)$.

Lemma 9. *There is a polynomial-time algorithm that, given an s^2-t^2 path P^2 in G^2, finds an s-t path P in G satisfying* $|\mathcal{L}(P)| \le |\mathcal{L}^2(P^2)|^{1/2}$.

Theorem 10. *For any fixed $k \ge 1$, MinLP cannot be approximated in polynomial time within a factor of $O(\log^k n)$ unless* P = NP.

Proof. The reduction described in Lemma 7 produces MinLP instances in which the underlying graph is planar. Therefore, the result stated in this lemma also applies to instances $I = (G, \mathcal{L}, s, t)$ in which G is an n-vertex planar graph. Now suppose that there exists a polynomial-time algorithm $\mathcal{A}$ whose approximation factor for such instances is $\alpha(n) \le c_k \ln^k n$, for some $c_k > 0$. We show that this algorithm can utilize the label squaring operation to obtain a self-improvability property, as a result of which we derive an approximation factor smaller than $c \ln n$ for planar MinLP, where c is the constant mentioned in Lemma 7.

We assume that n is sufficiently large so that $\ln^{1/2}(3n) \le \frac{c}{4} \ln n$, and let $q = q(k, c_k)$ be the smallest integer satisfying $c_k^{2^{-q}} \le 2$, $2^{2^{-q}qk} \le 2$ and $2^{-q}k \le \frac{1}{2}$. Such a constant indeed exists, since $c_k^{2^{-q}} \to 1$, $2^{2^{-q}qk} \to 1$, and $2^{-q}k \to 0$ as q tends to infinity. Starting with a planar instance I, we repeatedly apply the label squaring operation q times, to obtain $I^{2^q} = (G^{2^q}, \mathcal{L}^{2^q}, s^{2^q}, t^{2^q})$. We then employ the algorithm $\mathcal{A}$ to find an approximate

s^{2^q}-t^{2^q} path in G^{2^q}, and make use of Lemmas 8 and 9 to obtain an s-t path P in G such that

$$|\mathcal{L}(P)| \leq \left(\alpha\left(|V(G^{2^q})|\right) \cdot \mathrm{OPT}\left(I^{2^q}\right)\right)^{2^{-q}} \leq \alpha^{2^{-q}}\left(|V(G^{2^q})|\right) \cdot \mathrm{OPT}(I) \ .$$

To bound the approximation guarantee $\alpha^{2^{-q}}(|V(G^{2^q})|)$ in terms of n and c, we first claim that the number of vertices in G^{2^q} is at most $(3n)^{2^q}$. For this purpose, it can be easily verified that the label squaring operation preserves planarity, implying that the instances I^{2^j} we obtain throughout the sequence are planar, and in particular $|E(G^{2^j})| \leq 3|V(G^{2^j})|-6$. By combining this property with the observation that $|V(G^{2^{j+1}})| \leq |V(G^{2^j})| \cdot |E(G^{2^j})|$, we can inductively prove that $|E(G^{2^j})| \leq (3n)^{2^j}$ and $|V(G^{2^j})| \leq 3^{2^j-1}n^{2^j}$, with room to spare. It follows that the approximation factor we derive is at most

$$\alpha^{2^{-q}}\left(|V(G^{2^q})|\right) \leq c_k^{2^{-q}} \ln^{2^{-q}k}\left((3n)^{2^q}\right) = c_k^{2^{-q}} 2^{2^{-q}qk} \ln^{2^{-q}k}(3n) \leq 4\ln^{1/2}(3n) \leq c\ln n \ . \quad \square$$

References

1. A. A. Ageev and M. Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.
2. S. Arora. Personal communication, November 2005.
3. S. Arora and M. Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003.
4. A. Avidor and U. Zwick. Approximating MIN 2-SAT and MIN 3-SAT. *Theory of Computing Systems*, 38(3):329–345, 2005.
5. M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 294–304, 1993.
6. H. Broersma, X. Li, G. Woeginger, and S. Zhang. Paths and cycles in colored graphs. *Australasian Journal on Combinatorics*, 31:299–311, 2005.
7. T. Brüggemann, J. Monnot, and G. J. Woeginger. Local search for the minimum label spanning tree problem with bounded color classes. *Operations Research Letters*, 31(3):195–201, 2003.
8. R. D. Carr, S. Doddi, G. Konjevod, and M. V. Marathe. On the red-blue set cover problem. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 345–353, 2000.
9. R.-S. Chang and S.-J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.
10. I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–486, 2005.
11. O. Goldschmidt, D. S. Hochbaum, and G. Yu. A modified greedy heuristic for the set covering problem with improved worst case bound. *Information Processing Letters*, 48(6):305–310, 1993.
12. R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.
13. R. Hassin, J. Monnot, and D. Segev. Approximation algorithms and hardness results for labeled connectivity problems, 2006. Available at `http://www.math.tau.ac.il/~segevd`.
14. D. R. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

15. S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.
16. S. O. Krumke and H.-C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66(2):81–85, 1998.
17. D. H. Lorenz and D. Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters*, 28(5):213–219, 2001.
18. M. V. Marathe and S. S. Ravi. On approximation algorithms for the minimum satisfiability problem. *Information Processing Letters*, 58(1):23–29, 1996.
19. R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 475–484, 1997.
20. A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 588–597, 2001.
21. Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84(2):99–101, 2002.
22. H.-C. Wirth. *Multicriteria Approximation of Network Design and Network Upgrade Problems*. PhD thesis, Department of Computer Science, Würzburg University, 2001.
23. Y. Xiong, B. Golden, and E. Wasil. Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. *Operations Research Letters*, 33(1):77–80, 2005.

An Expressive Temporal Logic for Real Time

Yoram Hirshfeld and Alexander Rabinovich

School of Computer Science
Sackler Faculty of Exact Sciences
Tel Aviv University, Tel Aviv, Israel 69978
{joram, rabinoa}@post.tau.ac.il

Abstract. We add to the standard temporal logic with the modalities "Until" and "Since", a sequence of "counting modalities": For each n the modality $C_n(X)$, which says that X will be true at least at n points in the next unit of time, and its past counterpart $\overleftarrow{C}_n$, which says that X has happened at least n times in the last unit of time. We prove that this temporal logic is as expressive as can be hoped for; all the modalities that can be expressed in a strong natural decidable predicate logic framework, are expressible in this temporal logic.

1 Introduction

Temporal Logic based on the two modalities "Since" and "Until" (TL) is a popular among computer scientists as the framework for reasoning about a system evolving in time. By Kamp's theorem [18] this logic has the same expressive power as the first order monadic logic of order.

The two logics are (expressive) equivalent whether the system evolves in discrete steps or in continuous time, but for continuous time both logics can not express properties like: "X will happen within 1 unit of time". A natural metric modality say "X will happen exactly after one unit of time." Unfortunately, the extension of TL by this modality is undecidable. Over the years different decidable extensions of TL were suggested. The logic which was most extensively discussed was $MITL$ [2,1,10]. Other logics are described in [4,19,24]. We find the language QTL (quantitative temporal logic) which is presented in [13,14,15] more natural and convenient, and we will use it in the discussion. QTL has the two modalities "Until" and "Since", and two more modalities: $\diamondsuit_1 X$ - X will be true sometime within the next unit of time, and $\overleftarrow{\diamondsuit}_1 X$ - X was true sometime in the last unit of time.

We call these metric extensions of the pure temporal logic *the simple metric temporal logics*. They all have the same expressive power, which indicates that they capture a natural fragment of what can be said about the systems. This does not mean that they express all that needs to be said, and it was left to be determined whether these language are as expressive as can be hoped for, and if not, what needs to be added. Two important questions were not answered:

1. Is this logic expressive enough to express all the important properties about evolving systems?
2. If not, which modalities should we add?

Apparently A. Pnueli was the first to ask these questions, when he conjectured that the simple metric logics cannot express the requirement that X and then Y will both happen in the coming unit of time [2,24].

In [16] we proved Pnueli's conjecture, and we showed a sequence of modalities of the type that Pnueli suggested, such that no finite set of modalities can express all of them. Specifically: For every natural n we defined the "Pnueli modality" $Pnu_n(X_1, \ldots, X_n)$, which states that there is an increasing sequence $t_1, \ldots, t_n$ of points in the unit interval ahead such that t_i satisfies X_i. We also defined the weaker "Counting modalities" $C_n(X)$ which states that X will be true at least at n points in the unit interval ahead. We proved in [16] that:

- QTL (or $MITL$) with the added modalities $Pnu_1, \ldots, Pnu_n$ can not express the modality C_{n+1}
- No temporal logic with finitely many modalities can express all the modalities $C_n(X)$ for all natural numbers n.

This answers the first of the two questions above as negatively as can be imagined. It seemed to bring an end to the hope to extend Kamp's result, to define a simple temporal logic that extends plain temporal logic and is equivalent to a strong metric predicate logic.

In [12,14] we tried to identify the metric predicate logic that is best suited to deal with systems that evolve in time. A logic that is as expressive as possible, and yet simple to have a decidable validity and satisfiability problem. This logic can then serve to define modalities that will produce temporal logics that are decidable.

We started with the predicate logic that has also the +1 function alongside the order relation and the unary predicate variables. This language is too strong, as the +1 function allows for the encoding of Turing computations, and is clearly undecidable. We first identified the fragment of the predicate logic that corresponds to the temporal logic QTL (and to $MITL$). This is the "Quantified Monadic Logic of Order", **QMLO** that has atomic formulas $t = s$, $t < s$ and $X(t)$, is closed under Boolean connectors and first order quantifications, and under the "**metric quantifiers**": *If $\varphi(t)$ is a formula in QMLO with t its only free variable then $(\exists t)_{>t_0}^{<t_0+1}\varphi(t)$ is a formula of QMLO (in the free variable t_0).*

"Metric quantifiers" are just notations: $(\exists t)_{>t_0}^{<t_0+1}\varphi$ is shorthand for $(\exists t)[(t_0 < t < t_0 + 1) \wedge \varphi]$. This is a restricted form to use +1 function.

To extend the expressive power we then modified *QMLO* into *Q2MLO*; *Q2MLO* has the same atomic formulas as *QMLO* and is closed under the Boolean connectives, and first-order quantifiers, however the rule for metric quantifiers is changed to:

If $\varphi(t_0, t)$ is a formula in Q2MLO with t and t_0 its only free variable then $(\exists t)_{>t_0}^{<t_0+1}\varphi(t_0, t)$ and $(\exists t)_{>t_0-1}^{<t_0}\varphi(t_0, t)$ are formulas of Q2MLO.

At first glance the difference between *QMLO* and *Q2MLO* may look small, but it is actually very big. In particular for every (non metric) first order property φ we can state: "φ holds on some short interval (of length less than 1) that starts here".

In [12] we proved that *Q2MLO* has a decidable validity and satisfaction problem. Simple attempts to further modify *Q2MLO* like non trivial properties of the whole interval ahead, or permitting an additional third first-order variable to be free inside metric quantifier would make the logic as expressive as *FOMLO* with the function +1 (hence undecidable).

We demonstrated that *Q2MLO* is expressive decidable predicate logic in which modalities can be defined with the assurance that the resulting temporal logic is decidable. Therefore the second question above becomes:

> Can we add a nice (necessarily infinite) family of modalities to the pure temporal logic $TL(Until, Since)$ and obtain a temporal logic that is complete for *Q2MLO*?

Here we give the surprising answer:

Theorem: The pure temporal logic $TL(Until, Since)$ together with all the counting modalities C_n and $\overleftarrow{C}_n$ is expressively equivalent to *Q2MLO*.

The paper is divided as follows: In section 2 we recall the definitions and the previous results concerning the continuous time logics. Section 3 recalls the compositional method, which is a main tool in the proof of the theorem. In section 4 we prove the main theorem. Section 5 states further results.

2 Monadic Logic and Quantitative Temporal Logic

2.1 MLO - Monadic Logic of Order

We start with the standard definitions.

The monadic predicate logic of order - MLO has in its vocabulary *individual* (first order) variables $t_0, t_1, \ldots,$, monadic *predicate* names $X_0, X_1, \ldots,$, and one binary relation $<$ (the order).

The first order predicate language over this vocabulary is called the (first order) **Monadic Logic of Order (*FOMLO*)**. Note that since we are interested only in the first order language, $X_0, X_1, \ldots,$ is a chosen sequence of constant predicate names, and not of set variables.

In this work **a structure for** *FOMLO* is a tuple $M = \langle \mathbb{R}, <, P_1, \ldots, P_n \rangle$, where $\mathbb{R}$ is real line, with and $P_1, \cdots, P_n$, are one-place predicates (sets) that correspond to the predicate names in the logic.

As is common we will use the assigned formal names to refer to objects in the meta discussion. Thus we will write: $M \models \varphi[\tau_1, \ldots, \tau_k; P_1, \ldots, P_m]$ where M is a structure, φ a formula, $\tau_1, \cdots, \tau_k$ elements of M and $X_1, \ldots, X_m$ predicates in M, instead of the correct but tedious form:

$$M, \tau_1, \ldots, \tau_k; P_1, \ldots, P_m \models_{MLO} \varphi(t_1, \ldots, t_k; X_1, \ldots, X_m),$$

where $\tau_1, \ldots, \tau_k$ and $P_1, \cdots, P_m$ are names in the metalanguage for elements and predicates in M.

Everything in this paper remains true if we consider the class of structures whose domain is the non negative part of the real line, with 0 as first element.

In surveying the background, we will mention two more classes of structures: The class of Rational time, whose structures have the rational numbers as their domain, and the *Finite Variability class*, which is the of structures over the real line, such that every unary predicate changes its value only finitely often in any bounded interval of time.

2.2 Temporal Logics

Temporal logics use logical constructs called "modalities" to create a language that is free from variables and quantifiers:

The syntax of the Temporal Logic $TL(O_1^{(k_1)}, \ldots, O_n^{(k_n)}, \ldots)$ has in its vocabulary *monadic predicate names* $P_1, P_2, \ldots$ and a sequence of *modality names* with prescribed arity, $O_1^{(k_1)}, \ldots, O_n^{(k_n)}, \ldots$ (the arity notation is usually omitted). The formulas of this temporal logic are given by the grammar:

$$\varphi \ ::= True |\ P \mid \neg\varphi \mid \varphi \wedge \varphi \mid O^{(k)}(\varphi_1, \cdots, \varphi_k)$$

A temporal logic with a finite set of modalities is called a finite (base) temporal logic.

A structure for Temporal Logic, in this work, is the real line with monadic predicates $M = \langle \mathbb{R}, <, P_1, P_2, \ldots, P_n \rangle$, where the predicate P_i are those which are mentioned in the formulas of the logic. Every modality $O^{(k)}$ is interpreted in the structure M as an operator $O_M^{(k)} : [\mathbb{P}(A)]^k \to \mathbb{P}(A)$ which assigns "the set of points where $O^{(k)}[S_1, \ldots, S_k]$ holds" to the k-tuple $\langle S_1, \ldots, S_k \rangle \in \mathbb{P}(\mathbb{R})^k$. (Here $\mathbb{P}(\mathbb{R})$ denotes the set of all subsets of $\mathbb{R}$). Once every modality corresponds to an operator the semantics is defined by structural induction:

- for atomic formulas: $\langle M, t \rangle \models_{TL} P$ iff $t \in P$.
- for Boolean combinations the definition is the usual one.
- for $O^{(k)}(\varphi_1, \cdots, \varphi_k)$

  $$\langle M, t \rangle \models_{TL} O^{(k)}(\varphi_1, \cdots, \varphi_k) \quad \text{iff} \quad t \in O_M^{(k)}(A_{\varphi_1}, \cdots, A_{\varphi_k})$$

  where $A_\varphi = \{ \tau : \langle M, \tau \rangle \models_{TL} \varphi \}$ (we suppressed predicate parameters that may occur in the formulas).

For the modality to be of interest the operator $O^{(k)}$ should reflect some intended connection between the sets A_{φ_i} of points satisfying φ_i and the set of points $O[A_{\varphi_1}, \ldots, A_{\varphi_k}]$. The intended meaning is usually given by a formula in an appropriate predicate logic:

Truth Tables: A formula $\overline{O}(t_0, X_1, \ldots X_k)$ in the predicate logic L is a *Truth Table* for the modality $O^{(k)}$ if for every structure M

$$O_M(A_1, \ldots, A_k) = \{\tau \ : \ M \models_{MLO} \overline{O}[\tau, A_1, \ldots, A_k]\} \ .$$

The modalities *until* and *since* are most commonly used in temporal logic for computer science. They are defined through the following truth tables:

– The modality X **U** Y - "*X until Y*", is defined by

$$\psi(t_0, X, Y) \equiv \exists t_1 (t_0 < t_1 \wedge Y(t_1) \wedge \forall t (t_0 < t < t_1 \rightarrow X(t))).$$

– The modality X **S** Y - "*X since Y*", is defined by

$$\psi(t_0, X, Y) \equiv \exists t_1 (t_0 > t_1 \wedge Y(t_1) \wedge \forall t (t_1 < t < t_0 \rightarrow X(t))).$$

A central issue in this work is whether some temporal logic is equivalent to a fragment of predicate logic. We will define exactly what is meant by it:

Definition 1 (Expressive Equivalence). *Let L be a a fragment of predicate logic, and let TL be some temporal logic. Let $\mathcal{M}$ be a class of structures such that interprets both logics.*

1. *If for every formula $\varphi(t)$ of L with a single free variable there is a formula ϕ of TL, such that for every structure M in $\mathcal{M}$ and for every $t \in M$*

$$\langle M, t\rangle \models_{TL} \phi \quad \textit{iff} \quad M \models \varphi[t]$$

   then we say that TL is at least as expressive as L in the class $\mathcal{M}$.
2. *A similar condition says when L is at least as expressive as TL over $\mathcal{M}$.*
3. *If both conditions hold we say that TL and L are expressively equivalent over $\mathcal{M}$, or that they have the same expressive power.*

If the modalities of a temporal logic have truth tables in a predicate logic then the temporal logic is equivalent to a fragment of the predicate logic. Formally:

Proposition 1. *If every modality in the temporal logic TL has a truth table in the logic FOMLO then to every formula $\varphi(X_1, \ldots, X_n)$ of TL there corresponds effectively (and naturally) a formula $\overline{\varphi}(t_0, X_1, \ldots X_n)$ of FOMLO such that for every M, $\tau \in M$ and predicates $P_1, \ldots, P_n$*

$$\langle M, \tau, P_1, \ldots, P_n\rangle \models_{TL} \varphi \quad \text{iff} \quad \langle M, \tau, P_1, \ldots, P_n\rangle \models_{MLO} \overline{\varphi}\ .$$

In particular the temporal logic $TL($ **U** , **S** $)$ with the modalities "until" and "since" corresponds to a fragment of first-order *MLO*.

The two modalities **U** and **S** are also enough to express all the formulas of first-order *FOMLO* with one free variable:

Theorem 2. *([18,9]) The temporal logic $TL($ **U** , **S** $)$ is expressively complete for FOMLO over the two canonical structures: For every formula of FOMLO with at most one free variable, there is a formula of $TL($ **U** , **S** $)$, such that the two formulas are equivalent to each other, over the positive integers (discrete time) and over the real line (continuous time).*

2.3 The Simple Metric Logics: Quantitative Temporal Logic, and Quantitative Monadic Logic of Order

The logics *MLO* and $TL($ **U** , **S** $)$ are not suitable to deal with statements like "X will occur within one unit of time". It might be tempting to include in the monadic logic of order also a unary function symbol to denote the function $t+1$. This however makes the language undecidable. Moreover, seemingly weak fragments of this logic are undecidable. The corresponding modification of the temporal logic would add to the logic the modality $S(X)$ that holds at a point t if $t+1 \in X$. This also results in an undecidable logic. For the last 20 years languages that can express such properties and are decidable were proposed and investigated ([4,19,8,24,10,13,15]), and most notorious, logic $MITL$ [2,1,10]. We will use as a framework the *Quantitative Temporal Logic, QTL* which was introduced in [12,13,14]. All these logics are expressively equivalent [13]. QTL is defined as follows:

Definition 3 (Quantitative Temporal Logic). *QTL,* quantitative temporal logic *is the logic $TL($ **U** , **S** $)$ enhanced by the two modalities: $\Diamond_1 X$ and $\overleftarrow{\Diamond}_1 X$. These modalities are defined by the tables with free variable t_0:*

$$\Diamond_1 X : \qquad \exists t((t_0 < t < t_0 + 1) \wedge X(t)) \tag{3}$$

$$\overleftarrow{\Diamond}_1 X : \qquad \exists t((t - 1 < t < t_0) \wedge X(t)) \ . \tag{4}$$

The temporal logic QTL is complete for a natural fragment of the monadic logic of order, enriched with the $+1$ function:

Definition 4 (Quantitative Monadic Logic of Order). *$QMLO$* , quantitative monadic logic of order *is the predicate logic that has atomic formulas $t = s$, $t < s$ and $X(t)$, where X ranges over unary predicate names; it is closed under Boolean connectors and first order quantifications, and has the following rule for the "***metric quantifiers***":*

> If $\varphi(t)$ is a formula in $QMLO$ with t its only free variable and $m < n$ are integers then $(\exists t)^{<t_0+n}_{>t_0+m}\varphi(t)$ is a formulas of $QMLO$.

Recall that the metric quantifier $(\exists t)^{<t_0+n}_{>t_0+m}\varphi$ is shorthand for $\exists t(t_0 + m < t < t_0 + n \wedge \varphi)$.

Theorem 5. *([13,15]) The temporal logic QTL is expressively equivalent to QMLO over the full, or positive half, real line.*

2.4 The Limited Expressive Power of the Simple Metric Logics

There was no reason to believe that the simple metric logics like QTL have comprehensive expressive power. A. Pnueli raised this question, and he conjectured that the modality $Pnu_2(X,Y)$ is not expressible in $MITL$, where $Pnu_2(X,Y)$ says that X and then Y will be true at points in the next unit of time [2,24].

In [16] we proved Pnueli's conjecture, and we strengthened it significantly. To do this we defined for every natural n the "Pnueli modality" $Pnu_n(X_1,\ldots,X_n)$, which states that there is an increasing sequence $t_1,\ldots,t_n$ of points in the unit interval ahead such that t_i satisfies X_i. We also defined the weaker "Counting modalities" $C_n(X)$ which states that X is true at least at n points in the unit interval ahead. I.e, $C_n(X) = Pnu_n(X,\ldots,X)$. With these modalities we proved:

Theorem 6. *1. QTL (or MITL) with the added modalities $Pnu_1, \ldots, Pnu_n$ can not express the modality C_{n+1}.*

2. No temporal logic with finitely many modalities can express all the modalities $C_n(X)$ for all natural numbers n. Hence no finite temporal logic will suffice to express everything of interest.

2.5 The Predicate Metric Logic *Q2MLO*

In classical monadic logic there is a natural logic suitable to deal with evolving systems, the logic *Q2MLO* , which was introduced in [12] .

Definition 7. *Q2MLO is the predicate logic that has atomic formulas $t = s$, $t < s$ and $X(t)$, where X ranges over unary predicate names; it is closed under Boolean connectors and first order quantifications, and has the following rule for the "***metric quantifiers***":*

If $\varphi(t_0, t)$ is a formula in *Q2MLO* with free first-order variables in $\{t_0, t\}$ and $m < n$ are integers then $(\exists t)^{<t_0+n}_{>t_0+m}\varphi(t_0, t)$ is a formulas of *Q2MLO*.

In [12] it was shown that:

Theorem 8. *The validity and satisfiability problem is decidable for Q2MLO, over continuous time, whether we are interested in the class of models with finite variability, or in the class of all models.*

QMLO and *Q2MLO* are both decidable, and they look similar. But there are some major differences:

1. It is very easy to express in *Q2MLO* properties that are not expressible in *QMLO* . Thus for example Pnueli's modality $Pnu_n(X_1, \ldots, X_n)$ has *Q2MLO* truth table $(\exists t)^{<t_0+1}_{>t_0}(\exists t_1, \cdots, t_n)[(t_0 < t_1 < \cdots < t_n < t) \wedge (X_1(t_1) \wedge \cdots \wedge X_n(t_n))]$
2. *Q2MLO* seems strong enough to express all the decidable modalities that we found in the literature, and we have yet to see a natural formula in some decidable logic that cannot be expressed in *Q2MLO*.
3. *QMLO* is expressively equivalent to *QTL* that has four modalities only. *Q2MLO* is not equivalent to any temporal logic with a finite number of modalities.

3 Elements of Composition Method

Here we recall some elements of the compositional method. This method will play an important role in the proofs of Section 4.

The "compositional method" applies to the case where a structure is composed from simpler structures, and the theory of the composite structure can be reduced to the theory of its components. Ehrenfeucht used it in [6] and our proofs follow his work. The method was developed and used by Fefeman-Vaught [7], Shelah [22] and others (see e.g., surveys [11,23,20]). Here is an introduction to what we need:

The structures of the form $M = (A, <, P_1, \ldots, P_m)$, where $<$ is a linear order on a set A and $P_i \subseteq A$ are called m-labelled chains.

Two m-labelled chains M, M' are called k-equivalent (written: $M \equiv_k M'$) if $M \models \varphi \;\Leftrightarrow\; M' \models \varphi$ for every sentence φ of quantifier depth k. This is an equivalence relation between labelled chains; its equivalence classes are called k-types for the given signature with $<$ and m unary predicate symbols. Let us list some fundamental and well-known properties of k-types.

Proposition 9. *1. For every m and k there are only finitely many k-types of m-labelled chains.*

2. For each k-type τ there is a sentence (called "characteristic sentence") which defines τ (i.e., is satisfied by a labelled m-chain iff it belongs to τ). For given k and m a finite list of characteristic sentences for all the possible k-types can be computed. (We take the characteristic sentences as the canonical representations of k-types. Thus, for example, transforming a type into another type means to transform sentences.)

3. Each sentence φ is equivalent to a (finite) disjunction of characteristic sentences; moreover, this disjunction can be computed from φ.

Given m-labelled chains M_0, M_1 we write $M_0 + M_1$ for their concatenation (ordered sum); the domain of $M_0 + M_1$ is the union of the domains of M_0 and M_1 (we assume that these domains are disjoint), the interpretation of a unary predicates P is the union of its interpretation in M_0 and in M_1, all elements of M_0 are less than all elements of M_1 and if two elements are in M_i, then their order in $M_0 + M_1$ is the same as in M_i.

We need the following composition theorem for ordered sums:

Theorem 10 (Composition Theorem). *The k-types of m-labelled chains M_0, M_1 determine the k-type of the ordered sum $M_0 + M_1$:*

This theorem justifies the notation $\tau_1 + \tau_2$ for the k-type of an m-chain which is the sum of two m-chains of k-types τ_1 and τ_2.

Corollary 11. *For every formula $\varphi(s,t)$ with free first-order variables s and t there is a finite list of pairs of formulas $\tau_1(s,t), \sigma_1(s,t), \ldots, \tau_k(s,t), \sigma_k(s,t)$, such that for any labelled chain M and three points $p < q < r$:*

$$M \models \varphi(p,r) \text{ iff for some } i \leq k,\ M \models \tau_i(p,q) \text{ and } M \models \sigma_i(q,r).$$

Moreover, the list $\tau_i(s,t), \sigma_i(s,t)$ is computable from φ.

4 Completeness of the Counting Modalities

Let TLC be the temporal logic $TL(\mathbf{U}, \mathbf{S})$ together with all the counting modalities $C_n(X)$ and $\overleftarrow{C}_n(X)$. We start to prove that TLC is equivalent to

Q2MLO. First we state two lemmas, that improve our understanding of the logic *Q2MLO*. The first lemma allows us to replace *Q2MLO* by a large temporal logic, whose modalities are all the modalities that can be defined by *Q2MLO* formulas with some restriction.

Lemma 12. *Let Γ be the set of all modalities with truth table of the form $(\exists t)^{<t_0+n}_{>t_0+m}\phi(t_0,t,X_1,\dots,X_k)$ and $(\exists t)^{<t_0-m}_{>t_0-n}\phi(t_0,t,X_1,\dots,X_k)$, with $0 \le m < n$, and where ϕ is a formula of FOMLO with the free first-order variables in $\{t_0,t\}$. Then, TL(Γ, Until, Since) is expressively equivalent to Q2MLO.*

We omit the proof of Lemma 12 for lack of space; it proceeds by the induction of nesting of metric quantifiers. The second lemma shows that *Q2MLO* can be defined using only simple intervals of length 1 in the definition. It will be the first of three applications of the composition method.

Lemma 13. *For every Q2MLO formula ψ there exists an equivalent Q2MLO formula ϕ which uses only metric quantifiers of the form $(\exists t)^{<t_0+1}_{>t_0}$ and $(\exists t)^{<t_0}_{>t_0-1}$.*

Proof. (Sketch) We exemplify the method proving the following two statement. Let $\varphi(t,s)$ be a pure *FOMLO* formula.

1. if $n > 1$ then $(\exists s)^{<t_0+n}_{>t_0}\varphi(t_0,s)$ is equivalent to a formula which uses metric quantifiers $(\exists s)^{t+m}_{t}$ where $m < n$.
2. if $n > 0$ then $(\exists s)^{<t_0+n+1}_{>t_0+n}\varphi(t_0,s)$ is equivalent to a formula which uses metric quantifiers $(\exists s)^{t+n}_{t+n-1}$ and $(\exists s)^{t+1}_{t}$.

All other cases can be treated similarly, or proven from these cases by induction on the numbers that appears in the metric quantifiers together with the technique used in the proof of Lemma 12.

(1) By Corollary 11 there are formulas $\tau_1,\sigma_1,\dots,\tau_k,\sigma_k$, such that for any three points $p < q < r$, $\varphi(p,r)$ is true iff for some $i \le k$, $\tau_i(p,q)$ and $\sigma_i(q,r)$ hold. It follows that

$$(\exists s)^{<t_0+n+1}_{>t_0}\varphi(t_0,s) \equiv \bigvee_{i=1}^{k}(\exists q)^{<t_0+n}_{>t_0}(\tau_i(t_0,q) \wedge (\exists s)^{<q+1}_{>q}\sigma_i(q,s))$$

Indeed if the right side of the equivalence is true then for the q and s that satisfy it there is some i such that $\tau_i(t_0,q)$ and $\sigma_i(q,s)$ and therefore $\varphi(t_0,s)$. The distance from t_0 to q is less than n and the further distance to s is less than 1. Hence s testifies the left side of the equivalence. If on the other hand s testifies the property on the left, then if $t_0 + n < s$ we choose any q between $s-1$ and t_0+n. If $s \le t_0+n$ then we may choose any q smaller than s which is larger than $s-1$ (and t_0). This will satisfy the right side.

(2) We deal similarly with $(\exists s)^{<t_0+n+1}_{>t_0+n}\varphi(t_0,s)$: By the corollary to the compositional theorem there are pairs of formulas: $\sigma_1,\tau_1,\dots\sigma_k,\tau_k$, such that for every three points $t < p < s$ we have $\varphi(t,s)$ iff for some $i \le k$, $\tau_i(t,p)$ and $\sigma_i(p,s)$. Furthermore for every $i < k$ there are m_i pairs of formulas $\pi_{i,1},\eta_{i,1},\dots\pi_{i,m_i},\eta_{i,m_i}$

such that for every point q between p and s, $\sigma_i(p,s)$ holds iff for some $j \leq m_i$ both $\pi_{i,j}(p,q)$ and $\eta_{i,j}(q,s)$ hold. We will show that

$$(\exists s)^{<t_0+n+1}_{>t_0+n}\varphi(t_0,s) \equiv \bigvee_{i=1}^{k}(\exists p)^{<t_0+n}_{>t_0+n-1}[\tau_i(t_0,p) \wedge (\forall q)^{<p+1}_{>p}\bigvee_{j=1}^{m_i}(\pi_{i,j}(p,q) \wedge (\exists s)^{<q+1}_{>q}\eta_{i,j}(q,s))]$$

The expression on the right is already of the right form. Assume that there is some s that makes the left side true. Then we choose $p = s - 1$ and we will show that this p satisfies the right formula. Indeed for every q such that $p < q < p + 1 = s < q + 1$ the point s is the unit interval to the right of q, and since $\varphi(t_0, s)$ is true there is some $i < k$ for which $\tau_i(t_0, p)$ and $\sigma_i(p, s)$ hold. And since q lies between p and s, there is some $j < m_i$ for which $\pi_{i,j}(p, q)$ and $\eta_{i,j}(q, s)$ hold. Therefore the right statement is true. In the other direction, assume that the i-th disjunct of the right statement is true. Hence, there is some p in the interval $(t_0 + n - 1, t_0 + n)$ such that $\tau_i(t_0, p)$ is true and $q = t_0 + n$ is in the interval $(p, p + 1)$ and therefore there is s in the interval $(t_0 + n, t_0 + n + 1)$ and there is an index $j < m_i$ for which $\pi_{i,j}(p, q)$ and $\eta_{i,j}(q, s)$ is true. It follows from the main property of the formulas τ, π and η that $\varphi(t_0, s)$ is true. Since s is the interval $(t_0 + n, t_0 + n + 1)$, the left statement is true. □

Theorem 14 (Main theorem). *The temporal logic TLC and the monadic logic Q2MLO are expressively equivalent.*

Proof. (Sketch) In one direction, every formula of TLC is equivalent to a formula of *Q2MLO* as all its modalities have truth tables in *Q2MLO*.

For the other direction, by the last lemma, it suffices to prove that every *Q2MLO* formula ϕ with metric quantifiers of the form $(\exists s)^{<t+1}_{>t}$ and $(\exists s)^{<t}_{>t-1}$, can be expressed in TLC. To simplify the discussion, we avoid the careful yet cumbersome distinction between the free variables of a formula $\varphi(t, s)$ and their intended interpretation in the model. We will speak freely of "the interval (t, s)", and say that "the interval satisfies the formula φ", instead of "the interpretation satisfies $\varphi(t, s)$". We must show that $(\exists s)^{<t+1}_{>t}\varphi(t, s)$ can be expressed by a TLC formula, that holds at t. Following a technical inductive argument we may assume that φ is of pure monadic logic. We divide the proof into steps.

1. It maybe that $(\exists s)^{<t+1}_{>t}\varphi(t, s)$ holds because there is a sequence of points s to the right of t that converge to t, and such that $\varphi(t, s)$ holds. We call such t a φ-*limit.* This property can be written in pure "Until, Since" temporal logic. We may therefore look for a TLC formula that expresses $(\exists s)^{<t+1}_{>t}\varphi(t, s)$ under the assumption that t is not a φ-limit. We can then add the temporal formula that says that t is a φ-*limit*, as a disjunct.
2. An interval $[x, y]$ will be called a φ-*interval* if
   (a) either $\varphi(x, y)$ holds, or y is a limit of points z to the right, for which $\varphi(x, z)$ holds.
   (b) $[x, y]$ is minimal, in that no $x < z < y$ satisfies $\varphi(x, z)$.

   If $[x, y]$ is a φ-interval, then x will be called a *left* φ *point* and y is a *right* φ *point.* x is the *left* φ-*partner* of y, and y is the *right* φ-*partner* of x.

Every left φ point has a unique φ-interval and a unique right partner. (the dual does not hold for a right φ point).
Claim: Assume that $\varphi(x, y)$ holds, and x is not a φ-*limit.* Then:
(a) x is a left φ point, of some φ-interval.
(b) If $x < y < x + 1$ then $x < y' < x + 1$, where y' is the right φ partner for x .
(Just choose y' to be the greatest lower bound of the points y that satisfy $\varphi(x, y)$). Therefore from now on we replace $\varphi(t, s)$ in $(\exists s)^{<t+1}_{>t}\varphi(t, s)$ by ($[t, s]$ is a φ-interval). From now on when we say φ we mean this formula. It is not difficult to see that the properties : "x is a left φ point", "x is a right φ point" and "x, y is a φ pair", can be expressed in pure monadic logic of order.

3. The crucial property is that there is a simple computable bound on the length of a proper decreasing sequence of φ-intervals. We say that the interval $[x, y]$ is greater than $[x', y']$ if $x < x' < y' < y$. Let $\tau_1, \sigma_1, \ldots, \tau_k, \sigma_k$ as in the composition theorem 10, be the formulas such that for $x < z < y$, $\varphi(x, y)$ holds, iff one of the pairs $\sigma_i(x, z)$ and $\tau_i(z, y)$ hold. Then:
Claim: Every proper decreasing sequence of φ-intervals has depth at most k.
Proof: if $x_1 < \cdots < x_{k+1} < y_{k+1} < \cdots < y_1$ is such that $[x_i, y_i]$ is a φ interval and if z is in all these intervals then for every i there is some formula τ_j such that $[z, y_i]$ satisfies τ_j. By the pigeonhole principle there are some $y_i < y_{i'}$ such that both $[z, y_i]$ and $[z, y_{i'}]$ satisfy the same τ_j. Therefore both $[x_{i'}, y_i]$ and $[x_{i'}, y_{i'}]$ satisfy φ, which is impossible, as $[x_{i'}, y_{i'}]$ is a minimal such interval.
4. For every φ-interval $[t, s]$ we define its rank to be the length of the longest proper decreasing sequence of φ-intervals that starts with $[t, s]$. We denote by $\varphi_i(t, s)$ the claim that $[t, s]$ is a φ-interval of rank i, and by $r_i(s)$ the claim that s is the right endpoint of such an interval. Clearly:
(a) Every φ-interval has rank between 1 and k.
(b) The formulas $\varphi_i(x, y)$ and $r_i(s)$ are expressible in pure monadic logic of order.
5. We come to a third application of the composition theorem:
Claim: Let t be a point, and let i be some rank. Let $\tau_1, \sigma_1, \ldots, \tau_p, \sigma_p$ as in the composition theorem 10, be the formulas such that for $x < z < y$, $\varphi_i(x, y)$ holds, iff one of the pairs $\sigma_j(x, z)$ and $\tau_j(z, y)$ hold. Then only the first $p+1$ points to the right of t that satisfy $r_i(s)$ can be right endpoints of a φ-interval $[u, s]$ of rank i, such that $u \leq t$.
Proof: Assume that $s_1 < s_2 < \cdots < s_l$ is a sequence of points satisfying $r_i(s)$, and $t_1, \cdots, t_l$ are the corresponding left hand partners. Then $t_1 < \cdots < t_l$, or else one interval would contain another, and their rank would not be the same. It follows that all the right hand points of intervals of rank i that correspond to partners up to t come before those that correspond to partners greater than t. Moreover, there are at most $p + 1$ of them, or else there would be $j \neq j'$ such that $[t, s_j]$ and $[t, s_{j'}]$ that satisfy the same τ_v, and $[t_j, t]$ and $[t_{j'}, t]$ satisfy σ_v, therefore $[t_j, s_j]$ and $[t_j, s_{j'}]$ are φ-intervals of rank i with the same left endpoint which is impossible.

6. We may now collect the pieces: $(\exists s)_{>t}^{<t+1}\varphi_i(t,s)$ holds iff one of the following disjuncts for $j = 1, \ldots, p+1$ holds:
   There are at least j points between t and $t+1$ that satisfy $r_i(s)$ and t together with the j^{th} point among the points larger than t that satisfies $r_i(s)$, satisfy also $\varphi_i(t,s)$.
   Note that everything is pure monadic logic (and hence pure temporal), except the claim of the existence of j points with a pure monadic property, between t and $t+1$. Therefore this can be expressed by an TLC formula. It remains now to take the disjunction of these formulas,over the ranks (and add one disjunct for the case that t is a φ-limit, as in item 1).

5 Conclusion and Further Results

We added to the temporal logic $TL(Until, Since)$ all the modalities $C_n(X)$ - "X will be true at least at n points in the next unit of time", and $\overleftarrow{C}_n(X)$ -"X was true at least at n points in the last unit of time". The resulting temporal logic is complete for a strong, yet decidable monadic logic of order, *Q2MLO*.

The expressive completeness result can be extended to the rational time line. However, $TL(Until, Since)$ is not expressively complete for the *FOMLO* over the rational line. Stavi found two modalities $Until_{St}, Since_{St}$ which have the same expressive power over the class of all linear orders as the *FOMLO* [8]. We can prove that the Stavi modalities together with $C_n(X)$ and $\overleftarrow{C}_n(X)$ have the same expressive power over the rationals as *Q2MLO*.

As with the pure temporal logic $TL(Until, Since)$ there is a gap between the complexity (and succinctness) of the temporal logic and that of the corresponding predicate logic. In [17] we proved that the satisfiability problem for the temporal logic $TL(Until, Since, \{C_n, \overleftarrow{C}_n\}_{n=1}^{\infty})$ is $PSPACE$ complete.

References

1. R. Alur, T. Feder, T.A. Henzinger. The Benefits of Relaxing Punctuality. Journal of the ACM 43 (1996) 116-146.
2. R. Alur, T.A. Henzinger. Logics and Models of Real Time: a survey. In Real Time: Theory and Practice. Editors de Bakker et al. LNCS 600 (1992) 74-106.
3. J. A. Brzozowski, R. Knast. The dot depth hierarchy of star free languages is infinite. J. of Computing Systems Science 16 (1978) 37-55.
4. Baringer H. Barringer, R. Kuiper, A. Pnueli. A really abstract concurrent model and its temporal logic. Proceedings of the 13th annual symposium on principles of programing languages (1986), 173-183.
5. H.D. Ebbinghaus, J. Flum, Finite Model Theory. Perspectives in mathematical logic, Springer (1991).
6. A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. Fundamenta Mathematicae 49 (1961), pp. 129-141.
7. S. Feferman and R.L. Vaught. The first-order properties of products of algebraic systems. *Fundamenta Mathematicae* **47**:57–103, 1959.

8. D.M. Gabbay, I. Hodkinson, M. Reynolds. Temporal Logics volume 1. Clarendon Press, Oxford (1994).
9. D.M. Gabbay, A. Pnueli, S. Shelah, J. Stavi. On the Temporal Analysis of Fairness. 7th ACM Symposium on Principles of Programming Languages. Las Vegas (1980) 163-173.
10. T.A. Henzinger It's about time: real-time logics reviewed. in Concur 98, Lecture Notes in Computer Science 1466, 1998.
11. Y. Gurevich (1985). Monadic second-order theories. In *Model-Theoretic Logics*, (J. Barwise and S. Feferman, eds.), 479-506, Springer-Verlag.
12. Y. Hirshfeld and A. Rabinovich, A Framework for Decidable Metrical Logics. In Proc. 26th ICALP Colloquium, LNCS vol.1644, pp. 422-432, Springer Verlag, 1999.
13. Y. Hirshfeld and A. Rabinovich. Quantitative Temporal Logic. In Computer Science Logic 1999, LNCS vol. 1683, pp 172-187, Springer Verlag 1999.
14. Y. Hirshfeld and A. Rabinovich, Logics for Real Time: Decidability and Complexity. Fundam. Inform. 62(1): 1-28 (2004).
15. Y. Hirshfeld and A. Rabinovich, Timer formulas and decidable metric temporal logic. Information and Computation Vol 198(2), pp. 148-178, 2005.
16. Y. Hirshfeld and A. Rabinovich, Expressiveness of Metric Modalities for Continuous Time. In Proccedings of International Computer Science Symposium in Russia, LNCS vol. 3967, pp. 211-220, Springer Verlag, 2006.
17. Y. Hirshfeld and A. Rabinovich, On The complexity of the temporal logic with counting modalities. Manuscript, 2006.
18. H. Kamp. Tense Logic and the Theory of Linear Order. Ph.D. thesis, University of California L.A. (1968).
19. Z. Manna, A. Pnueli. Models for reactivity. Acta informatica 30 (1993) 609-678.
20. Makowsky, J. A. 2004. Algorithmic aspects of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic 126(1-3)*, 159–213.
21. A. Rabinovich. Expressive Power of Temporal Logics In Proc. 13th Int. Conf. on Concurrency Theory, vol. 2421 of Lecture Notes in Computer Science, pages 57–75. Springer, 2002.
22. S. Shelah. The monadic theory of order. *Ann. of Math.*, **102**, pp 349-419, 1975.
23. W. Thomas (1997). Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In *Structures in Logic and Computer Science: A Selection of Essays in Honor of A. Ehrenfeucht*, *Lecture Notes in Computer Science* **1261**:118-143, Springer-Verlag.
24. T. Wilke. Specifying Time State Sequences in Powerful Decidable Logics and Time Automata. In Formal Techniques in Real Time and Fault Tolerance Systems. LNCS 863 (1994), 694-715.

On Matroid Representability and Minor Problems

Petr Hliněný[1,2]

[1] Faculty of Informatics,
Masaryk University in Brno,
Botanická 68a, 602 00 Brno, Czech Republic
[2] Department of Computer Science,
VŠB – Technical University Ostrava,
17. listopadu 15, 708 33 Ostrava, Czech Republic
hlineny@fi.muni.cz

Abstract. In this paper we look at complexity aspects of the following problem (matroid representability) which seems to play an important role in structural matroid theory: Given a rational matrix representing the matroid M, the question is whether M can be represented also over another specific finite field. We prove this problem is hard, and so is the related problem of minor testing in rational matroids. The results hold even if we restrict to matroids of branch-width three.

Keywords: Matroid representability, minor, finite field, spike, swirl. 2000 Math subject classification: 05B35, 68Q17, 68R05.

1 Introduction

We postpone necessary formal definitions until later sections. Matroids present a wide combinatorial generalization of graphs. A useful geometric essence of a matroid is shown in its vector representation over a field $\mathbb{F}$; the elements–vectors of the representation can be viewed as points in the projective geometry over $\mathbb{F}$. Not all matroids, however, have vector representations. That is why the question of $\mathbb{F}$-representability of a matroid is important to solve.

Another motivation for our research lies in a current hot trend in structural matroid theory; work of Geelen, Gerards and Whittle, e.g. [4,5] extending significant portion of the Robertson–Seymour's Graph Minors project [15] to matroids. It turns out that matroids represented over finite fields $\mathbb{F}$ play a crucial role in that research, analogous to the role played by graphs embedded on surfaces in Graph Minors. Such a role is further justified by related works concerning logic and complexity aspects of matroids, e.g. our [8,10], and by a somehow surprising connection of binary matroids with graph rank-width [1] of Courcelle and Oum.

In this paper we prove that it is hard to decide whether a matroid given by a vector representation over the rational numbers, has a vector representation over a specific finite field $\mathbb{F}$ (Theorems 3.1 and 4.1). In particular this result implies that also the problem of minor testing in rational matroids is generally hard. We moreover prove that the minor testing problem is hard even for a certain small planar minor (Theorem 5.6).

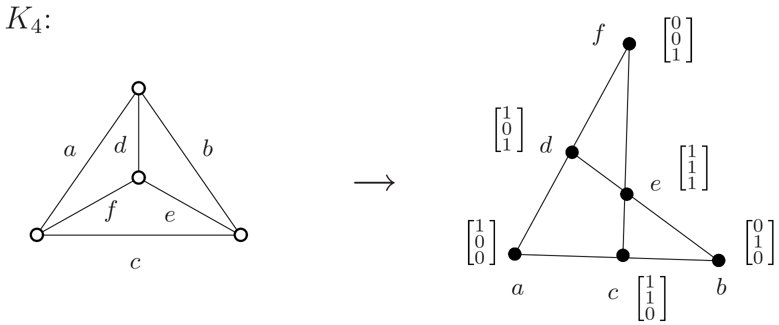
2 Matroids and Vector Representations

We refer to Oxley's book [12]. Since matroid theory seems not widely known among computer scientists, we should briefly review some basic terms here:

A *matroid* is a pair $M = (E, \mathcal{B})$ where $E = E(M)$ is the finite ground set of M (*elements* of M), and $\mathcal{B} \subseteq 2^E$ is a nonempty collection of *bases* of M, no two of which are in an inclusion. Moreover, matroid bases satisfy the "exchange axiom": if $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 \setminus B_2$, then there is $y \in B_2 \setminus B_1$ such that $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. Subsets of bases are called *independent sets*, and the remaining sets are *dependent*. Minimal dependent sets are called *circuits*. All bases have the same cardinality called the *rank* $\mathrm{r}(M)$ of the matroid. The *rank function* $r_M(X)$ in M assigns the maximal cardinality of an independent subset of a set $X \subseteq E(M)$. A set X is *spanning* if $r_M(X) = \mathrm{r}(M)$, and maximal non-spanning sets are called *hyperplanes*.

The reader may notice that a matroid, according to the presented definition, carries some information about all subsets of E which is exponential in the number of elements $|E|$. (See also [11].) Studying computational complexity on matroids one has to find a workaround for that: A common way to handle a matroid input is to consider a particular polynomially sized representation, instead.

If G is a (multi)graph, then its *cycle matroid* on the ground set $E(G)$ is denoted by $M(G)$: The independent sets of $M(G)$ are the acyclic subsets (forests) in G, and the circuits of $M(G)$ are the cycles in G. Another example of a matroid is a finite set of vectors with usual linear independence: If $\boldsymbol{A}$ is a matrix, then the


K_4:
a
d
b
f
e
c
$\longrightarrow$
f $\begin{bmatrix}0\\0\\1\end{bmatrix}$
d $\begin{bmatrix}1\\0\\1\end{bmatrix}$
e $\begin{bmatrix}1\\1\\1\end{bmatrix}$
a $\begin{bmatrix}1\\0\\0\end{bmatrix}$
b $\begin{bmatrix}0\\1\\0\end{bmatrix}$
c $\begin{bmatrix}1\\1\\0\end{bmatrix}$

$$M(K_4): \quad \boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Fig. 1. An example of a vector representation (on the right) $\boldsymbol{A}$ of the cycle matroid $M(K_4)$ of the complete graph K_4 (on the left). The matroid elements are depicted by dots, and their (linear) dependency is shown using lines.

matroid formed by the column vectors of $\boldsymbol{A}$ is called the *vector matroid* of $\boldsymbol{A}$, and denoted by $M(\boldsymbol{A})$. (Fig. 1.) The matrix $\boldsymbol{A}$ is a *vector representation* of a matroid $M \simeq M(\boldsymbol{A})$, and such $M(\boldsymbol{A})$ is $\mathbb{F}$-represented when $\boldsymbol{A}$ is over a field $\mathbb{F}$. We say that a matroid M is $\mathbb{F}$*-representable* if M has a vector representation over $\mathbb{F}$.

Questions of representability over finite fields $\mathbb{F}$ seem to play very important role in structural matroid theory. (We refer also to Section 5 for a closer discussion.) We now look at the problem from a complexity point of view.

Definition. The $\mathbb{F}$*-representability problem* for $\mathbb{F}'$-represented matroids is:

Input. A matrix $\boldsymbol{A}$ over a field $\mathbb{F}'$.
Question. Is the vector matroid $M(\boldsymbol{A})$ representable over $\mathbb{F}$?

Summary. If the input (matrix $\boldsymbol{A}$) is represented over $\mathbb{F}' = \mathbb{Q}$ (the rational numbers), the known answers to the $\mathbb{F}$-representability problem follow.

(2.1) For $\mathbb{F} = GF(2)$, the $\mathbb{F}$-representability problem for the matroid $M(\boldsymbol{A})$ is solvable in **polynomial** time by a deep result of Seymour [16].
(2.2) For $\mathbb{F} = GF(3)$, the answer is still open.
(2.3) For $\mathbb{F} = GF(q)$ a finite field on $q \geq 4$ elements, the $\mathbb{F}$-representability problem is ***co-NP*-complete** by our Theorems 3.1 and 4.1 and by [6].

Discussing (2.1), we explain that if a $GF(2)$-representable (*binary*) matroid M had also a vector representation over any field of characteristic not 2, then M could be represented by a totally-unimodular matrix. Hence M would be a so called regular matroid (representable over all fields), and then one can use Seymour's decomposition theorem [16] for regular matroids to recognize such $\mathbb{Q}$-represented M in polynomial time.

About (2.3) in the summary, one should understand why an apparently straightforward argument "guess an $\mathbb{F}$-representation and verify it" does not readily prove membership of the $\mathbb{F}$-representability problem in NP: The problem is that verifying whether two matrices represent isomorphic matroids may require evaluating too many subdeterminants. Indeed, Seymour [16] has proved that verifying a matrix over $GF(2)$ represents a matroid M (given by an oracle) requires testing independence on an exponential number of subsets. On the other hand, the following interesting result, showing membership of the $\mathbb{F}$-representability problem in *co*-NP, is proved in [6, Theorem 1.3]:

Theorem 2.4 (Geelen, Gerards and Whittle).
Let $\mathbb{F} = GF(q)$ be a finite field. Proving non-$\mathbb{F}$-representability of a matroid M needs only $O\big(|E(M)|^2\big)$ rank evaluations in M.

Lastly we add two remarks concerning possible extensions of our results in (2.3). First, the proofs of Theorems 3.1 and 4.1 show that the hardness result holds even for matroids of branch-width 3. On contrary to that, the $\mathbb{F}$-representability problem, as well as all other minor-closed properties, can be tested in polynomial time [7] if the input matroid is represented over a finite field $GF(q)$ and has bounded branch-width. Second, Theorems 3.1 and 4.1 could be extended to other infinite fields using the method of [9, Section 5]. We skip such extensions here to avoid the boring technical details.

3 Spikes: The Case of Non-prime Fields

The purpose of this section is to prove one case of the hardness result:

Theorem 3.1. *Let $\mathbb{F} = GF(q)$, where $q = p^a, a > 1$, be a finite non-prime field. Suppose $\boldsymbol{A}$ is a matrix given over $\mathbb{Q}$, and let $M = M(\boldsymbol{A})$ be its vector matroid. Then it is NP-hard to recognize that M has no vector representation over $\mathbb{F}$. The same conclusion remains true even if M is restricted to have branch-width 3.*

For the proof we need a definition of an interesting class of matroids, called "spikes". Let $n \geq 3$ and S_0 be a matroid circuit on the ground set $e_0, e_1, \ldots, e_n$. Denote by S_1 an arbitrary simple matroid obtained from S_0 by adding n new elements f_i for $i \in [1, n]$ such that $\{e_0, e_i, f_i\}$ is a triangle. Then the matroid $S = S_1 \setminus e_0$ obtained by deleting the central element e_0 is called a *rank-n spike.* The pairs $\{e_i, f_i\}$, $i \in [1, n]$ are called the *legs* of the spike. (Fig. 2.) Let *main circuits / bases* be those circuits / bases of S which intersect each leg of S in exactly one element. For instance, $\{e_1, \ldots, e_n\}$ is a main basis of S. We say that a spike S is a *free spike* if S has no main circuit. There is just one free spike of each rank up to isomorphism, see also Proposition 3.2(b),(e).

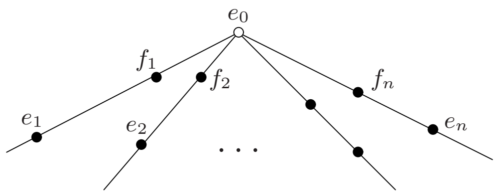


Fig. 2. An illustration of the definition of a rank-n spike

Spikes are known for giving "difficult counterexamples" in structural matroid theory. Some well known simple properties of spikes are summarized next; these implicitly originate perhaps in [13], and we refer to e.g. [9] for explicit proofs. Let $\boldsymbol{D}^1(x_1, \ldots, x_n) = [d_{i,j}]_{i=1}^n$ denote an $n \times n$ matrix such that $d_{i,j} = 1$ if $i \neq j \in [1, n]$, and $d_{i,i} = x_i$.

Proposition 3.2. *Let S be a rank-n spike where $n \geq 3$. Then*

a) the union of any two legs forms a 4-element circuit in S,
b) every other circuit intersects all legs of S,
c) S is 3-connected and the branch-width of S is 3,
d) S has a vector representation, if and only if it has a representation of the form $[\,\boldsymbol{I}_n \mid \boldsymbol{D}^1(x_1, \ldots, x_n)\,]$ where $x_1, \ldots, x_n \neq 1$ and $\boldsymbol{I}_n$ displays a chosen main basis of S, see e.g.

$$\begin{array}{c|cccccccccc}
 & e_1 & e_2 & \dots & e_{n-1} & e_n & f_1 & f_2 & \dots & f_{n-1} & f_n \\
\hline
e_1 & 1 & 0 & \cdots & 0 & 0 & x_1 & 1 & \cdots & 1 & 1 \\
e_2 & 0 & 1 & 0 & 0 & 0 & 1 & x_2 & 1 & 1 & 1 \\
\vdots & \vdots & 0 & \ddots & 0 & \vdots & \vdots & 1 & \ddots & 1 & \vdots \\
e_{n-1} & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & x_{n-1} & 1 \\
e_n & 0 & 0 & \cdots & 0 & 1 & 1 & 1 & \cdots & 1 & x_n
\end{array},$$

e) if S has a representation, as in (d), and $X \subseteq E(S)$ is such that X intersects each leg of S in exactly one element, then X is a circuit of S if and only if

$$\sum\nolimits_{j\in[1,n];\; f_j\in X} \frac{1}{x_j-1} = -1\,,$$

f) the free spike is $\mathbb{F}$-representable if $\mathbb{F}$ is a non-prime finite field.

Remark. Notice that a vector representation of a matroid in the "standard" form $\boldsymbol{A} = [\boldsymbol{I} \,|\, \boldsymbol{A}']$ has a property that the matroid bases are in a one-to-one correspondence with the nonzero subdeterminants in $\boldsymbol{A}'$ (via matrix pivoting).

Following [9], we have chosen to prove Theorem 3.1 via a polynomial reduction from the NP-complete *Partition* problem [2] over the integers. (Briefly saying, the *Partition* problem asks whether a given multiset of integers can be partitioned into two parts such that their sums equal.)

Let $\mathbb{F} = GF(q)$, where $q = p^a$, be a finite non-prime field.

Lemma 3.3. *If S is an $\mathbb{F}$-representable spike that is not the free spike, and $B \subset E(S)$ is any main basis of S, then there is a main circuit $C \subset E(S)$ such that $|C \setminus B| < q$ (independently of S).*

Proof. We refer to Proposition 3.2. According to (d), we select an $\mathbb{F}$-representation $[\,\boldsymbol{I}_n \,|\, \boldsymbol{D}^1(x_1,\dots,x_n)\,]$ of S where $\boldsymbol{I}_n$ shows the basis $B = \{e_1, e_2, \dots, e_n\}$. We choose the circuit C such that $|C \setminus B|$ is the smallest possible. Let $J = \{j \in [1,n];\; f_j \in C\}$. For a contradiction, we assume $|J| = |C \setminus B| \geq q$. Then among the partial sums $S(J') = \sum_{j\in J'} \frac{1}{x_j-1} \in \mathbb{F}$ for $J' \subseteq J$, there exist two such equal, $S(J_1) = S(J_2)$ where $J_1 \subset J_2 \subseteq J$, by the pigeon-hole principle. We set $J_0 = J_2 \setminus J_1$, and by (e) we may write

$$-1 = \sum\nolimits_{j\in J} \frac{1}{x_j-1} = S(J) = S(J) - S(J_2) + S(J_1) = \sum\nolimits_{j\in J\setminus J_0} \frac{1}{x_j-1}\,.$$

Hence there is another main circuit $C' = B\Delta\{e_j, f_j : j \in J \setminus J_0\}$ in S such that $|C' \setminus B| = |J \setminus J_0| < |J| = |C \setminus B|$, a contradiction to our choice of C. ∎

Proof of Theorem 3.1. Let $T = \{t_1, t_2, \dots, t_n\}$ be a multiset of positive integers – an input to the *Partition* problem, and $t = t_1 + t_2 + \ldots + t_n$. We denote by $z_i = \frac{-2t_i}{t}$ and set $x_i = \frac{1}{z_i} + 1$, for $i \in [1,n]$. We consider the matrix $\boldsymbol{A} = [\,\boldsymbol{I}_n \,|\boldsymbol{D}^1(x_1,\dots,x_n)\,]$ over $\mathbb{Q}$ as the input in the theorem. The vector matroid $S = M(\boldsymbol{A})$ is actually a spike by Proposition 3.2(d). Let $B = \{e_1,\dots,e_n\}$ be the main basis of S displayed by $\boldsymbol{I}_n$ in $\boldsymbol{A}$.

Assume $J \subset [1,n]$ is such that the multiset partition $\big(\{t_i : i \in J\}, \{t_i : i \in [1,n] \setminus J\}\big)$ is a solution to *Partition*. That is equivalent to $\sum_{i\in J} z_i = -1$, i.e. $\sum_{i\in J} \frac{1}{x_i - 1} = -1$. Hence by Proposition 3.2(e) the set $X = B \Delta \{e_i, f_i : i \in J\}$ is a main circuit. We conclude:

Claim 3.4. Solutions to the *Partition* problem on T are in a one-to-one correspondence with main circuits in the associated spike S.

Our polynomial reduction from *Partition* to $\mathbb{F}$-representability follows:

1. In the first stage we check by brute force all parts of T smaller than q. If we succeed, we answer YES to *Partition*.
2. In the second stage we ask about $\mathbb{F}$-representability of the matroid $S = M(\boldsymbol{A})$ defined above. If the outcome is negative, we again answer YES to *Partition*. Otherwise, our answer is NO.

It remains to prove that our reduction is correct. Assume the *Partition* problem on T has no solution (S is the free spike). Then in the first stage we find nothing, and in the second stage we answer NO by Proposition 3.2(f). Conversely, assume the *Partition* problem on T has a solution (T_1, T_2). If $\min(|T_1|, |T_2|) < q$, then we answer YES in stage 1. Otherwise, there is a main circuit C in S, but no such C with $|C \setminus B| < q$ by Claim 3.4. Hence S is not $\mathbb{F}$-representable by Lemma 3.3, and we answer YES in stage 2. ∎

4 Swirls: The Case of Prime Fields

Analogously to Theorem 3.1 we now finish the remaining, slightly more involved case of (2.3).

Theorem 4.1. *Let $\mathbb{F} = GF(p)$ be a finite prime field, $p \geq 5$. Suppose $\boldsymbol{A}$ is a matrix given over $\mathbb{Q}$, and let $M = M(\boldsymbol{A})$ be its vector matroid. Then it is NP-hard to recognize that M has no vector representation over $\mathbb{F}$. The same conclusion remains true even if M is restricted to have branch-width* 3.

For proving the statement we define another very interesting class of matroids called “swirls”, which have been implicitly introduced in [13]. Let rank-r *whirl*$\mathcal{W}^r$ be the unique matroid obtained from the cycle matroid of the wheel graph W_r with spokes $e_1, \dots, e_r$ and rim edges $f_1, \dots, f_r$, by relaxing (i.e. declaring independent) the rim circuit $\{f_1, \dots, f_r\}$. A simple $3n$-element matroid R is a rank-n *swirl* if R is obtained from the whirl $\mathcal{W}^n$ by adding a new element (denoted by g_i) on each dependent line (triangle $\{e_i, f_i, e_{i+1}\}$) of $\mathcal{W}^r$. (Fig. 3.)

The pairs $\{f_i, g_i\}$, $i \in [1,n]$ are called here *feet* of the swirl R. Let *main circuits / bases* be those circuits / bases of R which intersect each foot of R in exactly one element. For instance, $\{f_1, \dots, f_n\}$ is a main basis of S. The basis $\{e_1, \dots, e_n\}$ of R, formed by the spokes, is called the *central basis.* Notice that the central basis of a swirl is uniquely determined (unlike main bases). Swirls seem to be less known that spikes, and their structure is more complex. Fortunately, all the swirl properties we need in our proof have been implicitly proved in [13, Section 5], and we summarize them in the next proposition.

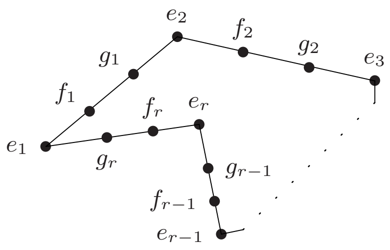


Fig. 3. An illustration of the definition of a rank-n swirl

Let $\boldsymbol{D}^2(x_1,\dots,x_n) = [d_{i,j}]_{i=1}^n$ denote an $n \times n$ matrix such that $d_{i,j} = 0$, $i,j \in [1,n]$ if $i \neq j, j+1$, $d_{i,i} = x_i$, and $d_{i+1,i} = 1$ (indices are take modulo n). Let the *free swirl* be a swirl having no main circuits. There is just one free swirl of each rank up to isomorphism, see also Proposition 4.2(a).

Proposition 4.2. *Let R be a rank-n swirl where $n \geq 4$. Then*

a) the only non-spanning circuits that possibly depend on a choice of elements g_i in the definition of a swirl are main circuits of R,
b) R is 3-connected and the branch-width of R is 3,
c) R has a vector representation, if and only if it has a representation of the form $[\,\boldsymbol{I}_n \,|\, \boldsymbol{D}^2(x_1,\dots,x_n) \,|\, \boldsymbol{D}^2(y_1,\dots,y_n)\,]$ where $x_1,\dots,x_n,y_1,\dots,y_n \neq 0$, $x_i \neq y_i$ and $x_1x_2\cdots x_n \neq (-1)^n$, $\boldsymbol{I}_n$ displays the central basis and $\{f_1,\dots,f_n\}$ displays any main basis of R, see e.g.

$$\begin{array}{cccccccccccccc}
e_1 & e_2 & \dots & e_{n-1} & e_n & f_1 & g_1 & f_2 & g_2 & & \dots & g_{n-1} & f_n & g_n \\
\hline
1 & 0 & \cdots & 0 & 0 & x_1 & y_1 & 0 & 0 & & \cdots & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & x_2 & y_2 & & 0 & 0 & 0 & 0 \\
 & & & 0 & 0 & 0 & 0 & 1 & 1 & & & 0 & 0 & 0 \\
\vdots & 0 & \ddots & & \vdots & \vdots & 0 & \vdots & 0 & & \ddots & 0 & \vdots & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & & y_{n-1} & 0 & 0 \\
0 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & & \cdots & 1 & x_n & y_n
\end{array},$$

d) if R has a representation, as in (c), and $X \subseteq E(R)$ is such that X intersects each foot of R in exactly one element, then X is a circuit of R if and only if

$$\prod\nolimits_{j\in[1,n];\, g_j\notin X} x_j \cdot \prod\nolimits_{j\in[1,n];\, g_j\in X} y_j = (-1)^n\,,$$

e) the free swirl is $\mathbb{F}$-representable if $\{1,-1\}$ is a proper 2-element subgroup of the multiplicative group $\mathbb{F}^*$ (in particular if $\mathbb{F} = GF(p)$ where $p \geq 5$).

Remark. Notice that, if two matroids agree on all non-spanning circuits, then they are isomorphic.

Looking at Proposition 4.2(d), one might get an easy idea: How about solving this case in direct analogy with Section 3, relating the main circuits of a swirl with solutions to a kind of a "product-partition" problem? This idea is generally good, but an immediate approach—to "lift" the *Partition* problem to exponents, fails since such a reduction produces exponentially large instances. We use the following intermediate step (reduced from the 3-dim matching problem):

Definition. The *ProdSelect* problem over the integers is:

Input. A multiset of k positive integers T, and an integer c.
Question. Is there a subset $P \subseteq T$ such that $\prod_{t \in P} t = c$?

Lemma 4.3. *ProdSelect is an NP-complete problem.*

Due to space restrictions, we skip some supplementary proofs. Let $\mathbb{F} = GF(p)$ be a prime field. Analogously to Lemma 3.3 one may prove:

Lemma 4.4. *If R is an $\mathbb{F}$-representable swirl that is not the free swirl, and $B \subset E(R)$ is any main basis of R, then there is a main circuit $C \subset E(R)$ such that $|C \setminus B| < p$ (independently of R).*

Proof of Theorem 4.1. Let $T = \{t_1, t_2, \ldots, t_{n-1}\}$ and c, positive integers, form an input to *ProdSelect* (NP-complete by Lemma 4.3). We may clearly assume $\min(T) > 1$, $c > 1$ and $c \notin T$. Let a matrix $\boldsymbol{A} = \left[\, \boldsymbol{I}_n \mid \boldsymbol{D}^2\big(1, \ldots, 1, (-1)^n c^{-1}\big) \,|\boldsymbol{D}^2\big(t_1, \ldots, t_{n-1}, (-1)^{n-1}\big) \,\right]$ over $\mathbb{Q}$ be the input in the theorem. The vector matroid $R = M(\boldsymbol{A})$ is actually a swirl by Proposition 4.2(c). Denote by $B = \{f_1, \ldots, f_n\}$ the main basis of R formed by the columns of $\boldsymbol{D}^2(1, \ldots, 1, (-1)^n c^{-1})$, cf. 4.2(d).

Assume $J \subseteq [1, n-1]$ is such that the multiset $P = \{t_i : i \in J\}$ is a solution to *ProdSelect*. That is equivalent to $(-1)^n c^{-1} \cdot \prod_{i \in J} t_i = (-1)^n$. So by Proposition 4.2(d) the set $X = B \varDelta \{f_i, g_i : i \in J\}$ is a main circuit. Notice also that our assumption $\min(T), c > 1$ implies that all main circuits X – solutions to the equation of 4.2(d), must involve term c^{-1}, in other words $g_n \notin X$. Hence:

Claim 4.5. Solutions to the *ProdSelect* problem on T, c are in a one-to-one correspondence with main circuits in the associated swirl R.

Our polynomial reduction from *ProdSelect* to $\mathbb{F}$-representability follows:

1. In the first stage we check by brute force all subsets of T smaller than p. If we succeed, we answer YES to *ProdSelect*.
2. In the second stage we ask about $\mathbb{F}$-representability of the matroid $R = M(\boldsymbol{A})$ defined above. If the outcome is negative, we again answer YES to *ProdSelect*. Otherwise, our answer is NO.

It remains to prove that our reduction is correct. Assume the *ProdSelect* problem on T has no solution (R is the free swirl). Then we answer NO by Proposition 4.2(e). Conversely, assume the *ProdSelect* problem on T has a solution P. If $|P| < p$, then we answer YES in stage 1. Otherwise, there is a main circuit C in R, but no such C with $|C \setminus B| < p$ by Claim 4.5. Hence R is not $\mathbb{F}$-representable by Lemma 4.4, and we answer YES in stage 2. ∎

The proof of (2.3) is now complete by Theorems 2.4, 3.1 and 4.1.

5 Matroid Minors

The Graph Minor project [14,15] of Robertson and Seymour is commonly considered a milestone in structural graph theory. Moreover, the project has had a great impact into theoretical computer science: We mention in particular an $O(n^3)$-time algorithm for testing whether an input graph contains a minor isomorphic to a fixed graph, implying efficient algorithmic solutions to all minor-closed graph properties.

In the more general setting of structural matroid theory, direct extensions of the great Graph Minors results are often false, but a stream of new theoretical results of Geelen, Gerards and Whittle, e.g. [4,5] in past several years extended significant portion of the Graph Minors theory to matroids representable over finite fields. Questions of matroid representability, and the notion of branch-width, turned out to be the key ingredients in that research. On the algorithmic side, that effort has been contributed by a sequence of results of the author dealing with FPT computation of matroid branch-width and recognition of MSO-definable matroid properties, e.g. [7,8,10].

Formally, a *minor* N of a matroid M is obtained by a sequence of deletions and contractions of elements, the order of which does not matter. The meaning of deletion is standard, and *contraction* is the dual operation to deletion, analogous to contraction of a graph edge. In geometric terms, a contraction M/e means a linear projection from the point representing e. We write $N = M \setminus D/C$ where D are the deleted and C the contracted elements.

Definition. The *N-minor problem* for represented matroids is such:

Input. A matrix $\boldsymbol{A}$ over a field $\mathbb{F}'$.
Question. Does the vector matroid $M(\boldsymbol{A})$ contain a minor isomorphic to a fixed matroid N?

Note that N may be arbitrary in the problem, but fixed; N is not considered a part of the input to the problem. The problem easily belongs to NP:

Lemma 5.1. *Let a matroid N be fixed. One needs only bounded number of rank evaluations to prove that a matroid M has a minor isomorphic to N.*

Proof. Let $M' = M \setminus D/C$ be a minor of M. The formula $\mathrm{r}_{M'}(X) = \mathrm{r}_M(X \cup C) - \mathrm{r}_M(C)$ determines the rank function of M' relatively to M (for example, [12, Chapter 3]). The minor M' is isomorphic to N if and only if $|E(M')| = |E(N)|$ and the rank functions equal for some bijection between $E(M')$ and $E(N)$. That can be verified, after guessing C and D, using only bounded number of rank evaluations in M. ∎

Obviously, for some very simple N such as the circuits $U_{k,k+1}$ the matroid N-minor problem is polynomial. We remark that $U_{r,n}$ denotes the matroid made of n points (vectors) in general position in rank r. It is a kind of a miracle that for $\mathbb{Q}$-represented matroids the $U_{2,4}$-minor problem is also polynomial. See (2.1) and Seymour [16]; the claim follows from the fact that $U_{2,4}$ is the only forbidden minor for binary matroids. On the other hand, since we know the seven forbidden minors [3] for matroid representability over the field $GF(4)$, it follows from Theorem 3.1 that the matroid N-minor problem is hard for at least some N. Such an argument, however, does not apply to minors N that are cycle matroids of graphs (*graphic* N), and hence representable over all fields.

It is proved in [7,8] that for every finite field $\mathbb{F}'$ one can solve the N-minor problem for $\mathbb{F}'$-represented matroids in polynomial time when restricted to inputs of bounded branch-width. Moreover using the matroid version of the "excluded grid" theorem [5], one can solve the N-minor problem for $\mathbb{F}'$-represented matroids in polynomial time, when N is a planar graphic matroid, regardless of branch-width. Therefore it is particularly interesting how difficult is the N-minor problem for $\mathbb{Q}$-represented matroids when N is a planar graphic matroid. We provide the answer in the rest of this section.

Summary. If the input (matrix $\boldsymbol{A}$) is represented over $\mathbb{F}'$, complexity cases of the N-minor problem follow:

(5.2) If $\mathbb{F}'$ is a finite field and the branch-width of $M(\boldsymbol{A})$ is bounded, then the problem is solvable in **cubic time** [7].

(5.3) If $\mathbb{F}'$ is a finite field and N is the cycle matroid of a planar graph, then the problem is solvable in cubic time [5,7], too.

(5.4) If $\mathbb{F}'$ is a finite field and N is arbitrary, then complexity of the N-minor problem is a very interesting open question in structural matroid theory.

(5.5) For $\mathbb{F}' = \mathbb{Q}$, the N-minor problem is ***NP*-complete**, even when the branch-width of $M(\boldsymbol{A})$ is three and N is the cycle matroid of a planar graph.

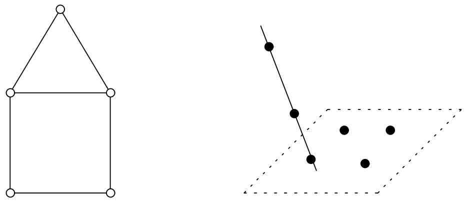


Fig. 4. The planar graph G_6, and its cycle matroid $M(G_6)$ on the right

Let G_6 denote the planar graph on 6 vertices formed by a 3-cycle and a 4-cycle sharing an edge, see Fig. 4. Our main result reads:

Theorem 5.6. *Let the planar graph G_6 be as in Fig. 4. Suppose $\boldsymbol{A}$ is a given matrix over $\mathbb{Q}$, and let $M = M(\boldsymbol{A})$ be its vector matroid. Then it is NP-complete to decide whether M has a minor isomorphic to the matroid $M(G_6)$. The same conclusion remains true even if M is restricted to have branch-width* 3.

We build on the following result of [9], which follows also from Claim 3.4.

Proposition 5.7. *Let S be the vector matroid of $[\,\boldsymbol{I}_n \mid \boldsymbol{D}^1(x_1,\dots,x_n)\,]$ over $\mathbb{Q}$ where $n \geq 5$. It is NP-hard to recognize that S is* not *the rank-n free spike.*

In order to use this statement in making a reduction for the matroid minor problem, we have to find a forbidden minor characterization of the free spikes. This is, after all, not so difficult.

Lemma 5.8. *Let S be the rank-n free spike. Then S has no $M(G_6)$-minor.*

Proof. Let $G_6 = C_3 \cup C_4$, where the two cycles C_3, C_4 share one edge. We use the same notation C_3, C_4 for the corresponding two circuits in the cycle matroid $M_6 = M(G_6)$ of G_6. Assume that $M_6 = S \setminus D/C$ is a minor of the free spike S, where C is the independent set of contracted elements of S. Let us call *leg circuits* in S the 4-element circuits formed by pairs of legs. Recall that main circuits are those circuits of S which intersect each leg in exactly one element. So S is the free spike, i.e. having no main circuits, if and only if all non-leg circuits in S are spanning (Proposition 3.2(b)). By the definition of a minor, a set X is a circuit in M_6 if and only if there is a circuit Y in S such that $X \subseteq Y$ and $Y \setminus X \subseteq C$.

Firstly, we claim that not both of the circuits C_3, C_4 of the minor M_6 result by contracting leg circuits in S. If that was not true, then there would be two leg circuits $L_1 \supseteq C_3$ and $L_2 \supseteq C_4$ in S. Since, actually, $|L_2| = 4 = |C_4|$, no element of L_2 was contracted or deleted when making M_6, and so $C_3 \cap C_4 = L_1 \cap L_2$. However, $|L_1 \cap L_2| \in \{0, 2\}$ for any two distinct leg circuits, but $|C_3 \cap C_4| = 1$, a contradiction.

Secondly, we consider that $C_4 \subset K$, where K is a spanning circuit in S, and $K \setminus C_4 \subseteq C$ (C are the contracted elements of S). So it is $|C| \geq |K| - |C_4| = n + 1 - 4 = n - 3$, and the rank $\mathrm{r}(M) = \mathrm{r}(S) - |C| \leq n - (n - 3) = 3$. However, $\mathrm{r}(M(G_6)) = 4 > 3$. The same contradiction turns out when the remaining possibility $C_3 \subset K$ is considered. Hence S cannot be the free spike, and the statement follows by means of contradiction. ∎

Lemma 5.9. *Let S be a rank-n spike for $n \geq 5$ that is not the free spike. Then S has an $M(G_6)$-minor.*

Proof. If S is not the free spike, then there is a main circuit $D \subset E(S)$, cf. Proposition 3.2(b). Let us use the notation from the definition of a spike. We form a minor N of S by contracting the elements of $\{e_i, f_i : i = 5, 6, \ldots, n\} \cap D$ and deleting the elements of $\{e_i, f_i : i = 2, 3, \ldots, n-1\} \setminus D$. Without loss of generality we assume $e_1, e_n \in D$. Then $\{e_1, f_1, f_n\}$ is a triangle in N and $\{e_i, f_i : i = 1, 2, 3, 4\} \cap D$ is a 4-element circuit in N, which intersect each other in e_1. Since there are no other non-spanning circuits in N, the minor N is isomorphic to $M(G_6)$. ∎

Proof of Theorem 5.6. If we could decide an $M(G_6)$-minor in the matroid $M = M(\boldsymbol{A})$ efficiently, then for a specific matrix $\boldsymbol{A} = [\,\boldsymbol{I}_n \mid \boldsymbol{D}^1(x_1, \ldots, x_n)\,]$, we would be able to decide whether $M(\boldsymbol{A})$, a spike by Proposition 3.2, is the free spike by Lemmas 5.8 and 5.9. Hence the statement and (5.5) are concluded by Proposition 5.7. ∎

Acknowledgments

The research has been supported by Czech research grant GAČR 201/05/0050, and partly by the program "Information Society" of the Czech Academy of Sciences, project No. 1ET101940420. The author would in particular like to express his thanks to an anonymous referee of the paper [9], who suggested to extend our complexity results (on spikes and swirls) in the direction towards the $\mathbb{F}$-representability problem.

References

1. B. Courcelle, S. Oum, *Vertex-Minors, Monadic Second-Order Logic, and a Conjecture by Seese*, J. Combin. Theory Ser. B, to appear (accepted 2006).
2. M.R. Garey, D.S. Johnson, Computers and Intractability, W.H. Freeman and Company, New York 1979.
3. J.F. Geelen, A.H.M. Gerards, A. Kapoor, *The excluded minors for GF*(4)*-representable matroids*, J. Combin. Theory Ser. B 79 (2000), 247–299.
4. J.F. Geelen, A.H.M. Gerards, G.P. Whittle, *Branch-Width and Well-Quasi-Ordering in Matroids and Graphs*, J. Combin. Theory Ser. B 84 (2002), 270–290.
5. J.F. Geelen, A.H.M. Gerards, G.P. Whittle, *Excluding a Planar Graph from a GF*(q)*-Representable Matroid*, manuscript, 2003.
6. J.F. Geelen, A.H.M. Gerards, G.P. Whittle, *Inequivalent Representations of Matroids I: An Overview*, in preparation, 2005.
7. P. Hliněný, *On Matroid Properties Definable in the MSO Logic*, In: Math Foundations of Computer Science MFCS 2003, Lecture Notes in Computer Science 2747, Springer Verlag Berlin (2003), 470–479.
8. P. Hliněný, *Branch-Width, Parse Trees, and Monadic Second-Order Logic for Matroids*, J. Combin. Theory Ser. B 96 (2006), 325–351.
9. P. Hliněný, *On Some Hard Problems on Matroid Spikes*, Theory of Computing Systems, to appear (accepted 2005).
10. P. Hliněný, D. Seese, *Trees, Grids, and MSO Decidability: from Graphs to Matroids*, Theoretical Computer Sci. 351 (2006), 372–393.
11. D. Mayhew, *Matroid Complexity and Large Inputs*, manuscript, 2005.
12. J.G. Oxley, Matroid Theory, Oxford University Press, 1992.
13. J.G. Oxley, D. Vertigan, G. Whittle, *On Inequivalent Representations of Matroids over Finite Fields*, J. Combin. Theory Ser. B 67 (1996), 325–343.
14. N. Robertson, P.D. Seymour, *Graph Minors – A Survey*, Surveys in Combinatorics, Cambridge Univ. Press 1985, 153–171.
15. N. Robertson, P.D. Seymour, *Graph Minors XX: Wagner's Conjecture*, J. Combin. Theory Ser. B 92 (2004), 325–357.
16. P.D. Seymour, *Decomposition of Regular Matroids*, J. Combin. Theory Ser. B 28 (1980), 305–359.

Non-cooperative Tree Creation
(Extended Abstract)

Martin Hoefer [*]

Department of Computer & Information Science,
Konstanz University, Box D 67, 78457 Konstanz, Germany
hoefer@inf.uni-konstanz.de

Abstract. In this paper we consider the *connection game*, a simple network design game with independent selfish agents that was introduced by Anshelevich et al [4]. In addition we present a generalization called *backbone game* to model hierarchical network and backbone link creation between existing network structures. In contrast to the connection game each player considers a number of groups of terminals and wants to connect at least one terminal from each group into a network. In both games we focus on an important subclass of *tree games*, in which every feasible network is guaranteed to be connected.

For tree connection games, in which every player holds 2 terminals, we show that there is a Nash equilibrium as cheap as the optimum network. We give a polynomial time algorithm to find a cheap $(2+\epsilon)$-approximate Nash equilibrium, which can be generalized to a cheap $(3.1+\epsilon)$-approximate Nash equilibrium for the case of any number of terminals per player. This improves the guarantee of the only previous algorithm for the problem [4], which returns a $(4.65+\epsilon)$-approximate Nash equilibrium. Tightness results for the analysis of all algorithms are derived.

For single source backbone games, in which each player wants to connect one group to a common source, there is a Nash equilibrium as cheap as the optimum network and a polynomial time algorithm to find a cheap $(1+\epsilon)$-approximate Nash equilibrium.

1 Introduction

Analyzing networks like the Internet, which is created and maintained by independent selfish agents with relatively limited goals, has become a research area in game theory and computer science attracting a lot of interest. In the considered models it is important to explore the boundary between stable and socially efficient solutions. Hence, it is of interest to characterize the price of stability [3], which is the ratio of the cost of the best Nash equilibrium over the cost of a socially optimum solution. This captures *how good stability can get*, and has been studied in routing and network creation games [3, 4, 10, 16]. The more prominent measure is the price of anarchy [12] describing the cost of the worst instead of the best Nash equilibrium. It has received attention in networking problems, for instance routing [15], facility location [17] and load balancing [6, 12]. In

[*] Supported by DFG Research Training Group 1042 "Explorative Analysis and Visualization of Large Information Spaces".

this paper we consider these measures for the *connection game*, a game-theoretic model for network topology creation introduced by Anshelevich et al [4]. In a connection game each of the k selfish agents has a connectivity requirement, i.e. she holds a number of terminals at some nodes in a given graph and wants to connect these nodes into a component. Possible edges have costs, and agents offer money to purchase them. Once the sum of all agents offers for an edge exceeds its cost, it is considered *bought*. Bought edges can be used by *all* agents to establish their connection, no matter whether they contribute to the cost. Each agent tries to fulfill her connection requirement at the least possible cost.

In the connection game it might be optimal for the agents to create disconnected local subnetworks. The Internet, however, receives its power as a platform for information sharing and electronic trade from the fact that it is *globally* connected. Hence, it is reasonable to assume that agents to some extent have an interest in being connected to the network of other agents. We incorporate this idea by focusing on *tree connection games* - connection games, in which every feasible solution is connected. Furthermore we study the interest in globally connected networks in hierarchical networks with an extended model, which we call *backbone game*. We assume a scenario with existing, globally unconnected subnetworks of small capacity. Each agent wants to connect a set of subnetworks with a connected network of high performance backbone links. Backbone links can start and end at any terminal in the subnetworks, so we can consider subnetworks as groups of terminals in the graph and adjust the connectivity requirements to be present between certain groups. Each player must connect at least one terminal of each of her groups into a connected network at the least cost. Purchasing and using edges works similar to the connection game.

Related Work. The connection game was introduced and studied in [4], where a variety of results were presented. Both prices of anarchy and stability are $\Theta(k)$. It is NP-complete to determine, whether a given game has a Nash equilibrium at all. There is a polynomial time algorithm that finds a $(4.65 + \epsilon)$-approximate Nash equilibrium on a 2-approximate network. For the single-source case, in which each player needs to connect a single terminal to a common source, a polynomial time algorithm finds a $(1 + \epsilon)$-approximate Nash equilibrium on a 1.55-approximate solution. We denote these algorithms by ADTW-SS for the single source and ADTW for the general case. In [3] adjusted connection games were used to study the performance of the Shapley value cost sharing protocol. Each edge is bought in equal shares by each player using it to connect its terminals. The price of stability in this game is $O(\log k)$. Furthermore extended results were presented, e.g. on delays, weighted games and best-response dynamics. Recently, connection games have been studied in a geometric setting. In [10] bounds were shown on the price of anarchy and the minimum incentives to deviate from an assignment purchasing the socially optimum network. The case of 2 players and 2 terminals per player was characterized in terms of prices of anarchy and stability, approximate equilibria and best-response dynamics.

A network creation game of different type was considered in [7,5,2]. Here each agent corresponds to a node and can only create edges that are incident to her node. Similar settings are recently receiving increased attention in the area of social network analysis. See [11] for a recent overview over developments in the area of social network design games. In the context of large-scale computational networks, however, a lot of these models lack properties like arbitrary cost sharing of edges and complex connectivity requirements.

Our Results. In this paper we will consider tree connection games (TCG) and single source backbone games (SBG). The games exhibit connection requirements such that every feasible solution network is connected. We analyze them with respect to strict and approximate pure-strategy Nash equilibria. We are especially interested in deterministic polynomial time algorithms for a two-parameter optimization problem: Try to assign payments to the players such that the purchased feasible network is cheap and the incentives to deviate are low. In Section 2 we show that for any tree connection game with two terminals per player the price of stability is 1. We outline an algorithm that allocates edge costs of a centralized optimum solution to players such that no player has an incentive to deviate. As this algorithm is not efficient, we show in Section 3 how to find a 2-approximate Nash equilibrium purchasing the optimum network for TCGs with any number of terminals per player. It can be translated into a polynomial time algorithm for $(3.1 + \epsilon)$-approximate Nash equilibria purchasing a network of cost at most 1.55 times the optimum network cost. This improves over ADTW that provides $(4.65 + \epsilon)$-approximate Nash equilibria. In addition we derive a tightness argument for the design technique of our algorithm and ADTW. Both algorithms consider only the optimum network and use a bounding argument for deriving approximate Nash equilibria. We show that both are optimal with respect to the class of deterministic algorithms working only on the optimum network. Thus, methods with better performance guarantees can be found, however, they must explicitly employ cost and structure of possible deviations. This significantly complicates their design and analysis.

In Section 4 we introduce the backbone game. Some results from the connection game translate directly: (1) both prices of anarchy and stability are in $\Theta(k)$, (2) it is NP-complete to determine, whether a given game has a Nash equilibrium and (3) there is a lower bound of $\left(\frac{3}{2} - \epsilon\right)$ on approximate Nash equilibria purchasing the optimum network. Here we show that for SBGs the price of stability is 1. A $(1 + \epsilon)$-approximate Nash equilibrium can be found in polynomial time. The procedure delivers the same results for three different generalizations: (1) games with a single source group, (2) games with a directed graph, in which players need a direct connection to the single source and (3) games, in which each player i has a threshold max(i) and would like to stay unconnected if the assigned cost exceeds max(i).

All proofs sketched or missing in this extended abstract will be given completely in the full version of this paper.

2 The Price of Stability

Connection Games. The connection game for k players is defined as follows. For each game there is an undirected graph $G = (V, E)$, and a nonnegative cost $c(e)$ associated with each edge $e \in E$. Each player owns a set of terminals located at nodes of the graph that she wants to connect. A strategy for a player i is a function p_i, which specifies for each edge the amount $p_i(e)$ that i offers for the purchase of e. If the sum of the offers of all players to an edge e exceeds $c(e)$, the edge is bought. Bought edges can be used by all players to connect their terminals, no matter whether they contribute to the edge costs or not. An (a-approximate) Nash equilibrium is a payment scheme such that no player can reduce $\sum_{e \in E} p_i(e)$ (by more than a factor of a) by unilaterally choosing a different strategy. Note that each player insists on connecting her terminals, and hence considers only such strategies as alternatives.

The problem of finding an optimum centralized network for all players and an optimum strategy for a single player are the classic network design problems of the Steiner forest [1, 9] and the Steiner tree [14], respectively. For the rest of this paper we will denote an optimum centralized network by T^*. The subtree of T^* that player i uses to connect her terminals is denoted by T^i.

Tree Connection Games. We will deal with an interesting class of connection games, the tree connection games (TCG), which are games with tree connection requirements.

Definition 1. *In a connection game there are tree connection requirements if for any two nodes v_1 and v_{l+1} carrying terminals, there is a sequence of players $i_1, \dots, i_l$ and nodes $v_2, \dots v_l$ such that player i_j has terminals at nodes v_j and v_{j+1} for $j = 1, \dots, l$.*

A TCG can be thought of as a splitting of a single global player into k players, which preserves the overall connection requirements. For the subclass of TCG with 2 terminals per player we will use the term path tree connection game (PTCG). A first observation is that the price of anarchy of the PTCG is k. This is straightforward with an instance consisting of two nodes and two parallel edges, where each node holds a terminal of each player. One edge e_1 has cost k, the other edge e_2 a cost of 1. If each player is assigned to purchase a share of 1 of e_1, the solution forms a Nash equilibrium.

Our algorithmic framework for deriving exact and approximate Nash equilibria is as follows. Until there is no player left, in each iteration it picks a player, assigns payments, removes the player and reduces the edge costs by the amount she paid. As candidates for this it considers leaf players.

Definition 2. *A player owns a* lonely terminal *t, if t is located at a node, where no terminal of another player is located. A player i in a TCG is a* leaf player*, if she owns a lonely terminal, and there is at most one node with a non-lonely terminal of i.*

Algorithmic Framework

```
1. $c^1(e) = c(e)$ for all $e \in E$.
2. For $iter \leftarrow 1$ to $k$
3.    $i$ is a leaf player if possible; otherwise an arbitrary player
4.    Determine $p_i$ using $c^{iter}$
5.    Set $c^{iter+1}(e) \leftarrow c^{iter}(e) - p_i(e)$ for all $e \in E$
6.    Remove $i$, contract edges of cost 0
```

Theorem 1. *The price of stability in the PTCG is 1.*

Proof. To prove our theorem we need the following technical lemma. Consider a game in which T^* contains all nodes from the graph G. There are two players h and i and a single source terminal at a node s shared by both players. Player i holds exactly one additional terminal.

Lemma 1. *In the described game there is a Nash equilibrium as cheap as T^*.*

Proof. (sketch) Algorithm 1 is used to construct a Nash equilibrium purchasing T^* for a game as described in the lemma. It considers edges in reverse BFS order from s with adjusted edge costs. Let T_e denote the part of T^* below an edge e and T_u the part below a node u, where $e \notin T_e$ but $u \in T_u$. When assigning the cost of e we use a cost function c' with $c'(e') = 0$ for $e' \in T^* \backslash T_e$ and $c'(e') = c(e')$ otherwise. A_i and A_h are the cheapest feasible deviation trees excluding e for players i and h, resp. We first focus on the question, whether p_h allows a cheaper deviation for h. In opposite to player i it is not trivial for player h to assume that all edges outside of T_e have cost 0. Consider a node u where multiple subtrees join. We know for each edge $e_1, e_2, e_3, \ldots$ below u that the tree $T_{e_j} + e_j$ is the optimum forest to connect the terminals of T_{e_j} to $T^* \backslash T_{e_j}$. But player h owns terminals in possibly *all* subtrees T_{e_j}. Is there a cheaper forest for h to connect her terminals in T_u to $T^* \backslash T_u$ than her calculated contribution?

Algorithm 1

```
1. For each edge $e$ in reverse BFS order
2.     Find cheapest deviations $A_h$ and $A_i$ for players $h$ and $i$ under $c'$ and
       given $p_h$ and $p_i$ on $T_e$.
3.     Assign $p_i(e) = \min(c(e),\ c'(A_i) - p_i(T_e))$.
4.     Assign $p_h(e) = \min(c(e) - p_i(e),\ c'(A_h) - p_h(T_e))$.
```

Lemma 2. *The payment function p_h constructed by Algorithm 1 allows no cheaper deviation for player h.*

Proof. Let the edges $e_1, \ldots, e_l$ be the edges directly below a node u in T_u. Assume the algorithm was able to assign payments that cover the costs of each $T_{e_j} + e_j$, and that u is the first node, at which $p_h(T_u)$ is not optimal for h.

We create a new cost function c'_h with $c'_h(e') = 0$ for $e' \in T^* \backslash T_u$ and $c'_h(e') = c(e') - p_i(e')$ otherwise. We will see that T^* is the optimum network under c'_h. Suppose the cheapest deviation tree A_h is cheaper than T^* under c'_h (i.e. the contribution of h to T^*). W.l.o.g. A_h includes all edges of cost 0, especially all edges purchased completely by i and all edges of T^* outside T_u. Let T_{e_j} be a tree that is not completely part of A_h. Consider for each terminal t of h located in T_{e_j} the path from t to u in A_h. We denote this set of paths by P_{e_j}. Let P'_{e_j} be the set of subpaths from P_{e_j} containing for every $P \in P_{e_j}$ the first part between the terminal of h and the first node $w \notin T_{e_j}$. This node always exists because $u \notin T_{e_j}$, and it is in T^*, because T^* covers all nodes in G. The network $A_{e_j} = \bigcup_{P \in P'_{e_j}} P$ was considered as a feasible deviation when constructing the payments for $T_{e_j} + e_j$, as it connects every terminal in T_{e_j} to a node of $T^* \backslash T_{e_j}$. Furthermore, the payments of i were the same, hence the cost of A_{e_j} was the same. Using the assumption that u is the first node, for which T_u is not optimal, we know that $c(A_{e_j}) \geq c(T_{e_j} + e_j)$. So after substituting A_{e_j} by T_{e_j} and e_j in A_h, the new network is at least as cheap as $T_{e_j} + e_j$. To show that this new network is also feasible, suppose we iteratively remove a path $P \in P'_{e_j}$. Now there might other terminals, whose connections to u use parts of P. The last node w of P is the first node of P outside of T_{e_j}, and it stays connected to u as P is the first part of a path to u. All other nodes of P are in T_{e_j} and will be connected by T_{e_j} and e_j. Hence, all terminals affected by the

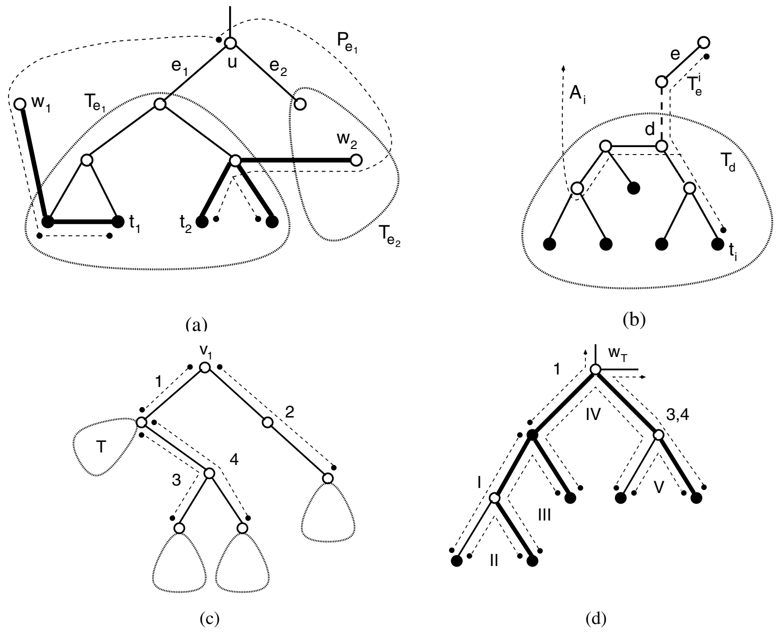


Fig. 1. (a), (b) Alternate trees and paths in PTCGs; (c) Distribution of hierarchical players for a parent player i, (d) distribution of personal players in the component T needed only by i

removal of P will finally be reconnected to u. In this way A_h can be transformed into T^* without cost increase. This proves that T^* is optimal under c'_h, so Algorithm 1 finds Nash equilibria. □

Figure 1a depicts the argument. Paths from the set P_{e_1} are indicated by dashed lines. The subgraph A_{e_1} of A_h is drawn bold. A node w can be either completely outside T_u (like w_1 for t_1) or in another T_{e_j} (like w_2 for t_2). Replacing A_{e_1} by T_{e_1} yields a feasible network that is not more expensive.

Finally, it is also possible to show that the payments calculated by Algorithm 1 purchase T^*. This proves Lemma 1. □

Now we outline how to use Lemma 1 to prove Theorem 1. Suppose we are given a PTCG with k players. At first we simplify the graph by constructing an equivalent *metric-closure game* with the same players, terminals and a complete graph G' on the nodes of T^*. Edge costs are equal to the cost of the shortest path in G. Then we use the framework with an induction on the number of players. Assume that the theorem holds for any PTCG with $k-1$ players. Now consider step 4 for a game with k players. If there is no leaf player, we can feasibly pick any player i and let $p_i = 0$. Otherwise, if i is a leaf player, we assign her to pay as much as possible on T^i such that she has no incentive to deviate. If the reduced network after the framework iteration is optimal under the reduced cost for the remaining $k-1$ players, the theorem follows with the induction hypothesis. Here we use Lemma 1 and Algorithm 1 to make the argument. Introduce a global player h, who accumulates all players except i. Players h and i share a single source at a node s, and an equilibrium assignment for h represents an optimal solution for the remaining players after removal of i. Hence, our inductive step is proven by Lemma 1. □

3 Approximation of Nash Equilibria

In this section we present an algorithm to calculate cheap approximate Nash equilibria in polynomial time. The algorithm sketched in the proof of Theorem 1 is not efficient. Either we must provide the socially optimum network as input or we must construct the optimum deviation for the collective player h to improve the solution network. In any case this requires to solve an instance of the Steiner tree problem. Instead, in this section we use *connection sets* to construct polynomial time approximation algorithms.

Definition 3. *[4] A connection set S of player i is a subset of edges of T^i, such that for each connected component C in $T^* \setminus S$ either (1) there is a terminal of i in C, or (2) any player that has a terminal in C has all of its terminals in C.*

For any node without a terminal and degree 2 in T^* the incident edges belong to the same connection sets, so for convenience we assume that T^* has no such nodes. If every player purchases at most α connection sets, the payments will form an α-approximate Nash equilibrium. Note that a subset of a connection set also is a connection set.

An algorithm for PTCGs. In connection games with 2 terminals per player the edges of T^* can be partitioned into equivalence classes S_J, where e and f belong to the same class iff $J = \{j : e \in T^j\} = \{j : f \in T^j\}$. Each S_J forms a connection set for all players $j \in J$, which is maximal under the subset relation. We will say that connection set S_J is *needed* by J. In the following PTCGs we will only consider maximal connection sets and not explicitly mention a player. This information is given by the subtrees, in which the set is located. Furthermore, when tree connection requirements are present, connection sets are contiguous.

Our algorithm uses the framework. In step 4 it assigns a leaf player i to purchase 2 connection sets. If i is no leaf player, $p_i = 0$. If a leaf player i is removed in step 5, distinct connection sets might join and subsequently be wrongly regarded as a single connection set. This happens when they are needed by player sets differing only by i. We will use the notion of endangered sets to refer to these problematic sets.

Definition 4. *A connection set is called* endangered set *for player i if it is needed by the set of players $J \cup \{i\}$, and there is another connection set (called* forcing set*) needed by the set J, with $i \notin J$.*

Lemma 3. *For any leaf player in a PTCG there are at most 2 endangered sets.*

We will denote the endangered set with empty forcing set as the *personal set*, the endangered set with nonempty forcing set as the *community set*. In step 4 we simply assign a leaf player to purchase these sets. It requires an easy inductive argument to show that the algorithm then works correctly. This yields the following theorem.

Theorem 2. *For any optimum centralized solution T^* in a PTCG, there exists a 2-approximate Nash equilibrium such that the purchased edges are exactly T^*.*

For PTCGs we can use a 1.55-approximation algorithm for the Steiner tree problem [14] to get an initial approximation T. Furthermore, we use shortest-path algorithms to find deviations and connection sets of optimum cost. Similar to [4] we can iteratively improve T by exchanging connection sets with better paths. Polynomial running time is ensured by substantial cost reduction in each exchange step, which can be achieved by an appropriate adjustment of the edge costs. Thus, for PTCGs there is an algorithm constructing $(2+\epsilon)$-approximate Nash equilibria on 1.55-approximate networks in time polynomial in n and ϵ^{-1}, for any $\epsilon > 0$.

An algorithm for TCGs. Next we adjust our algorithm to deliver 2-approximate Nash equilibria puchasing T^* for TCGs with any number of terminals per player. While this adjustment is possible using connection sets, it remains an open problem, whether Theorem 1 also holds for TCGs with more terminals per player.

Each player (denoted as parent player) is divided into a set of child players with 2 terminals per player. Terminals of the child players are located at the same nodes as the ones of the parent player. In addition terminals of child players are distributed such that they create a PTCG. Hence, the set of all child players can purchase T^* with 2 connection sets per player.

The algorithm again uses the framework, and in step 4 a special procedure to assign the cost of T^* to the parent player i. First player i is divided into child players. Then child players of i are iteratively assigned to purchase endangered sets and removed. In the end i has to purchase all edges assigned to her child players. To identify personal and community sets for the child players of i, one might first create a PTCG by splitting all other parent players. However, the next lemma ensures that the assignment of personal and community sets for a leaf parent player i does not depend on the splitting of the other parent players. In step 4 we thus assign edges without explicitly splitting other players than i.

Lemma 4. *The endangered sets of child players of a leaf parent player i are independent of the division of other parent players.*

In the remainder of the section we show how to divide a player i into *hierarchical*, *personal* and *superfluous* child players such that the union of connection sets purchased by her child players forms 2 connection sets. At first we disregard all non-lonely terminals but one. Denote the node carrying the last remaining non-lonely terminal t by v_1. If the player has only lonely terminals, we pick t arbitrarily. Then consider T^* rooted at v_1. Once we arrive at an edge e needed only by i, the tree connection requirements guarantee that the whole subtree below e is also needed only by i. Here we insert child players in a hierarchical fashion. We contract all edges that are needed only by i. Let this adjusted tree be denoted T' and consider it in BFS-order rooted at v_1. For each node v carrying a terminal of i, we introduce a new child player. She has a terminal at v and the nearest ancestor of v in the tree carrying a terminal of i (see Figure 1c). These child players will be termed *hierarchical players*.

Second, we consider the portions of the tree that were contracted to form T'. For each maximal connected subtree $T \subset T^i$ that is needed only by i, let w_T be the root node that T shares with T'. Let player j be the first hierarchical child player, whose terminal t_j was placed at w_T. This player connects upwards in T'. Now we consider

T in DFS-order and locate t_j at the first node carrying a terminal of i. For each new node w_z carrying a terminal encountered in the DFS order, we introduce a new child player and locate her terminals at the nodes w_{z-1} and w_z. At any time there is only one lonely terminal in T. Finally, when the last node w_l carrying a terminal of i is reached, we move all remaining terminals at w_T to w_l. They belong to the hierarchical players connecting downwards in T'. Child players introduced in the DFS-scan of the components T are called *personal players*, because they divide parts needed only by i (see Figure 1d).

Third, for every non-lonely terminal of i disregarded in the beginning, we introduce a *superfluous* child player connecting the terminal to v_1, which will not be assigned any payments.

Theorem 3. *For any optimum centralized solution T^* in a TCG, there exists a 2-approximate Nash equilibrium such that the purchased edges are exactly T^*.*

In the proof a special elimination order of child players is used. In any iteration a leaf player is picked, first the superfluous players and then in a bottom-up fashion to v_1 the personal and hierarchical players. One connection set for the parent player is formed by the union of all personal sets for the child players. The other connection set is the union of the community sets. In the full version we will show how to use polynomial time approximation algorithms to derive a $(3.1 + \epsilon)$-approximate Nash equilibrium on a 1.55-approximate network.

A tightness argument. For every connection game ADTW finds 3-approximate Nash equilibria purchasing T^* given only the optimum network T^* as input. Like our algorithm it uses connection sets and does not employ cost sharing of edges. The next theorem shows that no deterministic algorithm using only T^* as input can improve the guarantees even if it uses cost sharing. In this way ADTW for general connection games and our algorithm for TCGs represent optimal algorithms, but our algorithm provides a better guarantee on TCGs.

Theorem 4. *For any $\epsilon > 0$ there is a connection game [TCG] such that any deterministic algorithm using only T^* as input constructs a payment function, which is at least a $(3 - \epsilon)$ $[(2 - \epsilon)]$ approximate Nash equilibrium.*

Theorem 5. *For any $\epsilon > 0$ there is a TCG such that ADTW constructs a $(3 - \epsilon)$-approximate Nash equilibrium.*

4 Backbone Games

In this section we present the *backbone game*, an extension of the connection game to groups of terminals. Each of the k players has a set of groups of terminals. Each terminal may be located at a different node. The player strives to connect at least one terminal from each of her groups into a connected network. Different terminals may be located at the same nodes. Some important results from [4] translate directly to the backbone game by restriction to the connection game. The price of anarchy is k, and the price of stability $k - 2$. It is *NP*-complete to decide, whether a given game has a Nash equilibrium, and there is a lower bound of $\left(\frac{3}{2} - \epsilon\right)$ on approximate Nash equilibria purchasing T^*.

Finding the optimum network for a single player is the network design problem of the Group Steiner Tree (GSTP) [13]. The problem of finding a centralized optimum solution network T^* generalizes the GSTP in terms of forest connection requirements, so we term this the *Group Steiner Forest Problem (GSFP)*. There are polylogarithmic approximation algorithms for the GSTP [8], but we are not aware of any such results for the GSFP. Hence, we will concentrate on algorithms for games, in which the solution is guaranteed to be connected. The general case represents an interesting field for future work.

Single Source Backbones. In a SBG each player i has a group $\mathcal{G}_i$ of g_i terminals and must connect at least one terminal to a given source node s. Note that the price of anarchy is still k as the example establishing the bound is a single source game. However, the price of stability is 1, and cheap approximate equilibria can be found in polynomial time.

Theorem 6. *The price of stability in the SBG is 1. There is a polynomial time algorithm to find a $(1 + \epsilon)$-approximate Nash equilibrium purchasing a network T with $c(T)/c(T^*) \in O(\log n \log k \log(\max_i g_i))$.*

References

1. A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM J Comp*, 24(3):445–456, 1995.
2. S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty. On nash equilibria for a network creation game. In *Proc 17th Ann ACM-SIAM Symp Discrete Algorithms (SODA)*, pages 89–98, 2006.
3. E. Anshelevich, A. Dasgupta, J. Kleinberg, É. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Proc 45th Ann IEEE Symp Foundations Comp Sci (FOCS)*, pages 295–304, 2004.
4. E. Anshelevich, A. Dasgupta, É. Tardos, and T. Wexler. Near-optimal network design with selfish agents. In *Proc 35th Ann ACM Symp Theo Comp (STOC)*, pages 511–520, 2003.
5. J. Corbo and D. Parkes. The price of selfish behavior in bilateral network formation. In *Proc 24th Ann ACM Symp Principles of Distributed Comp (PODC)*, 2005.
6. A. Czumaj, P. Krysta, and B. Vöcking. Selfish traffic allocation for server farms. In *Proc 34th Ann ACM Symp Theory Comp (STOC)*, pages 287–296, 2002.
7. A. Fabrikant, A. Luthera, E. Maneva, C. Papadimitriou, and S. Shenker. On a network creation game. In *Proc 22nd Ann ACM Symp Principles of Distributed Comp (PODC)*, pages 347–351, 2003.
8. N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the Group Steiner tree problem. *J Algorithms*, 37:66–84, 2000.
9. M. Goemams and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J Comp*, 24(2):296–317, 1995.
10. M. Hoefer and P. Krysta. Geometric network design with selfish agents. In *Proc 11th Conf on Comp and Comb (COCOON), LNCS 3595*, pages 167–178, 2005.
11. M. Jackson. A survey of models of network formation: Stability and efficiency. In G. Demange and M. Wooders, editors, *Group Formation in Economics; Networks, Clubs and Coalitions*, chapter 1. Cambridge University Press, Cambridge, 2004.
12. E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proc 16th Ann Symp Theoretical Aspects Comp Sci (STACS)*, pages 404–413, 1999.

13. G. Reich and P. Widmayer. Beyond Steiner's problem: A VLSI oriented generalization. In *Proc 15th Intl Workshop on Graph-Theoretic Concepts Comp Sci (WG), LNCS 411*, pages 196–210, 1989.
14. G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proc 10th Ann ACM-SIAM Symp Discrete Algorithms (SODA)*, pages 770–779, 2000.
15. T. Roughgarden and É.Tardos. How bad is selfish routing? *J ACM*, 49(2):236–259, 2002.
16. A. Schulz and N. Stier Moses. Selfish routing in capacitated networks. *Math Oper Res*, 29(4):961–976, 2004.
17. A. Vetta. Nash equilibria in competitive societies with application to facility location, traffic routing and auctions. In *Proc 43rd Ann IEEE Symp Foundations Comp Sci (FOCS)*, page 416, 2002.

Guarantees for the Success Frequency of an Algorithm for Finding Dodgson-Election Winners

Christopher M. Homan[1] and Lane A. Hemaspaandra[2,⋆]

[1] Rochester Institute of Technology, Rochester, NY 14623, USA
[2] University of Rochester, Rochester, NY 14627, USA

Abstract. Dodgson's election system elegantly satisfies the Condorcet criterion. However, determining the winner of a Dodgson election is known to be Θ_2^p-complete ([1], see also [2]), which implies that unless $\mathrm{P} = \mathrm{NP}$ no polynomial-time solution to this problem exists, and unless the polynomial hierarchy collapses to NP the problem is not even in NP. Nonetheless, we prove that when the number of voters is much greater than the number of candidates (although the number of voters may still be polynomial in the number of candidates), a simple greedy algorithm very frequently finds the Dodgson winners in such a way that it "knows" that it has found them, and furthermore the algorithm never incorrectly declares a nonwinner to be a winner.

1 Introduction

The *Condorcet paradox* [3], otherwise known as *the paradox of voting* or *the Condorcet effect*, says that rational (i.e., well-ordered) individual preferences can lead to irrational (i.e., cyclical) majority preferences. It is a well-known and widely studied problem in the field of social choice theory [4]. A voting system is said to obey the *Condorcet criterion* [3] if whenever there is a Condorcet winner—a candidate who in each pairwise subcontest gets a strict majority of the votes—that candidate is selected by the voting system as the overall winner.

The mathematician Charles Dodgson (who wrote fiction under the now more famous name of Lewis Carroll) devised a voting system [5] that has many lovely properties and meets the Condorcet criterion. In Dodgson's system, each voter strictly ranks (i.e., no ties allowed) all candidates in the election. If a Condorcet winner exists, he or she wins the Dodgson election. If no Condorcet winner exists, Dodgson's approach is to take as winners all candidates that are "closest" to being Condorcet winners, with closest being in terms of the fewest changes to the votes needed to make the candidate a Condorcet winner. We will in Section 2 describe what exactly Dodgson means by "fewest changes," but intuitively speaking, it is the smallest number of sequential switches between adjacent entries in the rankings the voters provide. It can thus be seen as a sort of "edit distance."

Dodgson wrote about his voting system only in an unpublished pamphlet on the conduct of elections [5] and may never have intended for it to be published. It was eventually discovered and disseminated by Black [6] and is now regarded as a classic of

⋆ Supported in part by grant NSF-CCF-0426761 and the AvH TransCoop program.

social choice theory [4]. Dodgson's system was one of the first to satisfy the Condorcet criterion.[1]

Although Dodgson's system has many nice properties, it also poses a serious computational worry: The problem of checking whether a certain number of changes suffices to make a given candidate the Condorcet winner is NP-complete [2], and the problem of computing an overall winner, as well as the related problem of checking whether a given candidate is at least as close as another given candidate to being a Dodgson winner, is complete for Θ_2^p [1], the class of problems solvable with polynomial-time parallel access to an NP oracle [9]. (More recent work has shown that some other important election systems are complete for Θ_2^p: Hemaspaandra, Spakowski, and Vogel [10] have shown Θ_2^p-completeness for the winner problem in Kemeny elections, and Rothe, Spakowski, and Vogel [11] have shown Θ_2^p-completeness for the winner problem in Young elections.) The above complexity-theoretic results about Dodgson elections show, quite dramatically, that unless the polynomial hierarchy collapses there is no efficient (i.e., polynomial-time) algorithm that is guaranteed to always determine the winners of a Dodgson election. Does this then mean that Dodgson's widely studied and highly regarded voting system is all but unusable?

It turns out that if a small degree of uncertainty is tolerated, then there is a simple, polynomial-time algorithm, `GreedyWinner` (the name's appropriateness will later become clear), that takes as input a Dodgson election and a candidate from the election and outputs an element in $\{\text{"yes"}, \text{"no"}\} \times \{\text{"definitely"}, \text{"maybe"}\}$. The first component of the output is the algorithm's guess as to whether the input candidate was a winner of the input election. The second output component indicates the algorithm's confidence in its guess. Regarding the accuracy of `GreedyWinner` we have the following results.

Theorem 1. *1. For each (election, candidate) pair it holds that if `GreedyWinner` outputs "definitely" as its second output component, then its first output component correctly answers the question, "Is the input candidate a Dodgson winner of the input election?"*

2. For each $m,n \in \mathbb{N}^+$, the probability that a Dodgson election E selected uniformly at random from all Dodgson elections having m candidates and n votes (i.e., all $(m!)^n$ Dodgson elections having m candidates and n votes have the same likelihood of being selected[2]) has the property that there exists at least one candidate c such that `GreedyWinner` on input (E,c) outputs "maybe" as its second output component is less than $2(m^2-m)e^{\frac{-n}{8m^2}}$.

[1] The Condorcet criterion may at first glance seem easy to satisfy, but Nanson showed [7] that many well-known voting systems—such as the rank-order system [8] widely attributed to Borda (which Condorcet himself studied [3] in the same paper in which he introduced the Condorcet criterion), in which voters assign values to each candidate and the one receiving the largest (or smallest) aggregate value wins—fail to satisfy the Condorcet criterion.

[2] Since Dodgson voting is not sensitive to the *names* of candidates, we will throughout this paper always tacitly assume that all *m*-candidate elections have the fixed candidate set $1,2,\ldots,m$ (though in some examples we for clarity will use other names, such as *a*, *b*, *c*, and *d*). So, though we to be consistent with earlier papers on Dodgson elections allow the candidate set "*C*" to be part of the input, in fact we view this as being instantly coerced into the candidate set $1,2,\ldots,m$. And we similarly view voter *names* as uninteresting.

Thus, for elections where the number of voters greatly exceeds the number of candidates (though the former could still be within a (superquadratic) polynomial of the latter, consistently with the success probability for a family of election draws thus-related in voter-candidate cardinality going asymptotically to 1), if one randomly chooses an election $E = (C,V)$, then with high likelihood it will hold that for each $c \in C$ the efficient algorithm `GreedyWinner` when run on input (C,V,c) correctly determines whether c is a Dodgson winner of E, and moreover will "know" that it got those answers right. We call `GreedyWinner` a *frequently self-knowingly correct* heuristic. (Though the `GreedyWinner` algorithm on its surface is about *recognizing* Dodgson winners, as discussed in Section 3 our algorithm can be easily modified into one that is about, given an $E = (C,V)$, *finding* the complete set of Dodgson Winners and that does so in a way that is, in essentially the same high frequency as for `GreedyWinner`, self-knowingly correct.) Later in this paper, we will introduce another frequently self-knowingly correct heuristic, called `GreedyScore`, for calculating the Dodgson score of a given candidate.

The study of greedy algorithms has an extensive history (see the textbook [12] and the references therein). Much is known in terms of settings where a greedy algorithm provides a polynomial-time approximation, for example [13], and of guarantees that a greedy algorithm will frequently solve a problem—for example Kaporis, Kirousis, and Lalas study how frequently a greedy algorithm finds a satisfying assignment to a boolean formula [14] (and many additional excellent examples exist, e.g., [15, 16, 17, 18]). However, each of these differs from our work in either not being about *self-knowing* correctness or, if about self-knowing correctness, in being about an NP-certificate type of problem. In contrast, as discussed in more detail immediately after Definition 1, the problem studied in this paper involves objects that are computationally more demanding than mere certificates for NP set membership. (Additionally, our paper of course differs from previous papers in the problem itself, namely, we study Dodgson elections.)

Among earlier algorithms having a self-knowingly correct flavor, a particularly interesting one (though, admittedly, it is about finding certificates for NP set membership) is due to Goldberg and Marchetti-Spaccamela [19]. Goldberg and Marchetti-Spaccamela construct for every $\varepsilon > 0$ a modified greedy algorithm that is deterministic, runs in polynomial time (where the polynomial depends on ε), and with probability at least $1 - \varepsilon$ self-knowingly finds an optimal solution of a randomly chosen (from a particular, but natural, distribution) instance of the knapsack problem (they call their algorithm "self checking" rather than "self-knowingly correct"). Besides being about an NP-type problem rather than a "Θ_2^p"-type problem such as Dodgson elections, a second way in which this differs from our work is that the running time of our algorithms does not depend on the desired likeliness of correctness.

The concept of a heuristic that is effective on a significant portion of the problem instances of some very hard (i.e., strictly harder that NP-complete, unless NP = coNP) problem and furthermore has the additional property that it can very frequently guarantee that its answers are correct is related to other theoretical frameworks. Parameterized complexity [20] studies hard problems that have efficient solutions when instances of the problem are restricted by fixing one or more of the parameters that describe the instances. The two most natural parameters of Dodgson elections are the number of candidates and the number of voters. It is known that with either of these parameters bounded by a constant, Dodgson elections have polynomial-time algorithms [2]. However, we are interested in cases when no such dramatic bounding of a parameter by a constant occurs.

Like average-case [21] and smoothed [22] analyses, our analysis of `DodgsonWinner` is probabilistic. But while those methods are concerned with the expected value over segments of the problem domain of some resource of interest, typically time, we focus on a quite different property: correctness.

Like approximation algorithms, our algorithms are time efficient, even in the worst case. But approximation algorithms typically have worst-case guarantees on how far the answers they provide deviate from the optimal answers. We by contrast are only interested in how frequently the algorithms are correct. Even when we study optimization problems, as with `GreedyScore`, we make no guarantees on how close actual Dodgson scores are to the corresponding answers that `GreedyScore` provides in cases when the confidence is "maybe."

Additionally, the key feature that separates our work from each of the above-mentioned related frameworks is the "self-knowing" aspect of our work. The closest related concepts to this aspect of our analysis are probably those involving proofs to be verified, such as NP certificates and the proofs in interactive proof systems. Although our algorithms do not provide actual certificates, one could reasonably require a transcript of the execution of either of our algorithms. With such a transcript, it would be easy to verify in deterministic polynomial time that the algorithm (in cases where the confidence is "definitely") presented a valid proof. By contrast, in interactive proof systems the methods for verifying the proofs involve a probabilistic, interactive process. We do not consider interactive processes in this paper. In Section 3 we discuss in more detail the differences between heuristics that find NP certificates and self-knowingly correct algorithms, but the most obvious difference is that, unless coNP = NP, the problem of verifying whether a given candidate is a Dodgson winner for a given election is not even in NP.

Due to space limitations, all proofs are omitted, as are the conclusions, open problems, and some discussions and references. All of these can be found online in the TR version (at `xxx.arXiv.org` as revised report cs.DS/0509061).

2 Dodgson Elections

As mentioned in the introduction, in Dodgson's voting system each voter strictly ranks the candidates in order of preference. Formally speaking, for $m,n \in \mathbb{N}^+$ (throughout this paper we by definition do not admit as valid elections with zero candidates or zero voters), a *Dodgson election* is an ordered pair (C,V) where C is a set $\{c_1,\ldots,c_m\}$ of candidates (as noted earlier, we without loss of generality view them as being named by 1, 2, $\ldots$, m) and V is a tuple $(v_1,\ldots,v_n)$ of *votes* and a *Dodgson triple*, denoted (C,V,c), is a Dodgson election (C,V) together with a candidate $c \in C$. Each vote is one of the $m!$ total orderings over the candidates, i.e., it is a complete, transitive, and antireflexive relation over the set of candidates. We will sometimes denote a vote by listing the candidates in increasing order, e.g., (x,y,z) is a vote over the candidate set $\{x,y,z\}$ in which y is preferred to x and z is preferred to (x and) y. (Note: A candidate is never preferred to him- or herself.) For vote v and candidates $c,d \in C$, "$c <_v d$" means "in vote v, d is preferred to c" and "$c \prec_v d$" means "$c <_v d$ and there is no e such that $c <_v e <_v d$." Each Dodgson election gives rise to $\binom{m}{2}$ *pairwise races*, each of which is created by choosing two distinct candidates $c,d \in C$ and restricting each vote v to the two chosen candidates, that is, to either (c,d) or (d,c). The winner of the pairwise race is the one that a strict majority of voters prefer. Due to ties, a winner may not always exist in pairwise races.

A *Condorcet winner* is any candidate c that, against each remaining candidate, is preferred by a strict majority of voters. For a given election (i.e., for a given sequence of votes), it is possible that no Condorcet winner exists. However, when one does exist, it is unique.

For any vote v and any $c,d \in C$, if $c \prec_v d$, let $Swap_{c,d}(v)$ denote the vote v', where v' is the same total ordering of C as v except that $d <_{v'} c$ (note that this implies $d \prec_{v'} c$). If $c \not\prec_v d$ then $Swap_{c,d}(v)$ is undefined. In effect, a swap causes c and d to "switch places," but only if c and d are adjacent. The *Dodgson score* of a Dodgson triple (C,V,c) is the minimum number of swaps that, applied sequentially to the votes in V, make V a sequence of votes in which c is the Condorcet winner. A *Dodgson winner* is a candidate that has the smallest Dodgson score. This is the election system developed in the year 1876 by Dodgson (Lewis Carroll) [5], and as noted earlier it gives victory to the candidate(s) who are "closest" to being Condorcet winners. Note that if no candidate is a Condorcet winner, then two or more candidates may tie, in which case all tying candidates are Dodgson winners.

Several examples show how Dodgson elections work, and hint at why they are hard, in general, to solve: Consider an election having four candidates $\{a,b,c,d\}$ and one hundred votes in which sixty are (a,b,c,d) and forty are (c,d,a,b). Since d is already (i.e., before any exchanges take place) a Condorcet winner, d's Dodgson score is 0. Thus d is the Dodgson winner.

Now suppose in another election having the same candidates and the same number of voters as in the previous example that twenty voters each vote (a,b,c,d), (b,c,d,a), (c,d,a,b), (b,a,d,c), and (d,a,b,c), respectively. In this case, there is no Condorcet winner, so we calculate the Dodgson score of each candidate. Consider a: Candidate a beats d by twenty votes (of course, changes in the votes of as few as eleven voters can reverse such a shortfall), loses to c by twenty votes, and loses to b by twenty votes. What is the fewest number of swaps needed to make a a Condorcet winner? It might tempting to make, for each of eleven votes of the form (a,b,c,d), the following transformation: $(a,b,c,d) \rightarrow (d,b,c,a)$. But this transformation is not an allowed swap because only elements that are adjacent in the ordering imposed by the vote may be swapped. We can, however, make eleven of the following series of two swaps each: $(a,b,c,d) \rightarrow (b,a,c,d) \rightarrow (b,c,a,d)$. This can clearly be seen to be an optimal number of swaps in light of a's initial vote shortfalls (and note that every swap improves a's standing against either b or c in a way that directly reduces a still-existing shortfall), so the Dodgson score of (C,V,a) is twenty-two.

What makes it hard to calculate Dodgson scores is what makes many combinatorial optimization problems hard: There is no apparent, simple way to locally determine whether a swap will lead to an optimally short sequence that makes the candidate of interest the Condorcet winner. For instance, if in calculating the Dodgson score of a we had come across votes of the form (b,a,d,c) first, we might have made the following series of swaps, $(b,a,d,c) \rightarrow (b,d,a,c) \rightarrow (b,d,c,a)$, which contain a swap between a and d. But this series of swaps is not optimal because a already beats d, and because, as we saw, there is already a series of twenty-two swaps available, where each swap helps a against some adversary that a has not yet beaten, that makes a a Condorcet winner. (However, there are instances of Dodgson elections in which the only way for a candidate to become a Condorcet winner is for it to swap with adversaries that the candidate is already beating, so one cannot simply ignore this possibility.) Assuming that we did not at first see the votes that constitute this optimal sequence and instead hastily made swaps that did not affect a's current standing against b or c, we could have, as soon as we had come across votes of the form (a,b,c,d), backtracked from the hastily made swaps that led toward a nonoptimal solution and, eventually, have correctly calculated the Dodgson score. But as the number of candidates and votes in an election increases, the amount of backtracking that a naive strategy might need to make in order to correct for any nonoptimal swaps explodes.

Of course, computational complexity theory can give evidence of hardness that is probably more satisfying than are mere examples, such as those just given. However, before turning to the computational complexity of Dodgson-election-related problems, a couple of preliminary definitions are in order. The class NP is precisely the set of all languages solvable in nondeterministic polynomial time. Θ_2^p can be equivalently defined either as the class of languages solvable in deterministic polynomial time with $\mathcal{O}(\log n)$ queries to an NP language or as the class of languages solvable in deterministic polynomial time via parallel access to NP. Θ_2^p was first studied by Papadimitriou and Zachos in the 1980s, received its current name from Wagner, and has proven important in many contexts. In particular, it seems central in understanding the complexity of election systems [1,23,24,25,26,11]. All NP languages are in Θ_2^p. It remains open whether Θ_2^p contains languages that are not in NP; it does exactly if NP $\neq$ coNP.

Bartholdi, Tovey, and Trick [2] define the following decision problems, i.e., mappings from Σ^* to {"yes","no"}, associated with Dodgson elections. We here mostly copy the problem wordings from [1].

Decision Problem: `DodgsonScore`
Instance: A Dodgson triple (C,V,c); a positive integer k.
Question Is $Score(C,V,c)$, the Dodgson score of candidate c in the election specified by (C,V), less than or equal to k?

Decision Problem: `DodgsonWinner`
Input: A Dodgson triple (C,V,c).
Question: Is c a winner of the election? That is, does c tie-or-defeat all other candidates in the election?

Bartholdi, Tovey, and Trick show that the problem of checking whether a certain number of changes suffices to make a given candidate the Condorcet winner is NP-complete and that the problem of determining whether a given candidate is a Dodgson winner is NP-hard [2]. Hemaspaandra, Hemaspaandra, and Rothe show [1] that this latter problem, as well as the related problem of checking whether a given candidate is at least as close as another given candidate to being a Dodgson winner, is complete for Θ_2^p. Hemaspaandra, Hemaspaandra, and Rothe's results show that determining a Dodgson winner is not even in NP unless the polynomial hierarchy collapses. This line of work has significance because the hundred-year-old problem of deciding if a given candidate is a Dodgson winner was much more naturally conceived than the problems—arguably artificial—that were previously known to be complete for Θ_2^p (see [27]).

3 The `GreedyScore` and `GreedyWinner` Algorithms

In this section, we study the greedy algorithms `GreedyScore` and `GreedyWinner`, stated as, respectively, Algorithm 3.1 and Algorithm 3.2, and we note that their running time is polynomial. We show that both algorithms are self-knowingly correct in the sense of the following definition.

Definition 1. For sets S and T and function $f : S \to T$, an algorithm $\mathscr{A} : S \to T \times \{\text{"definitely"}, \text{"maybe"}\}$ is *self-knowingly correct for f* if, for all $s \in S$ and $t \in T$, whenever $\mathscr{A}$ on input s outputs $(t, \text{"definitely"})$ it holds that $f(s) = t$.

The reader may wonder whether "self-knowing correctness" is so easily added to heuristic schemes as to be uninteresting to study. After all, if one has a heuristic for finding certificates for an NP problem with respect to some fixed certificate scheme (in the standard sense of NP certificate schemes)—e.g., for trying to find a satisfying assignment to an input (unquantified) propositional boolean formula—then one can use the P-time checker associated with the problem to "filter" the answers one finds, and can put the label "definitely" on only those outputs that are indeed certificates. However, the problem studied in this paper does not seem amenable to such after-the-fact addition of self-knowingness, as in this paper we are dealing with heuristics that are seeking objects that are computationally much more complex than mere certificates related to NP problems. In particular, a polynomial-time function-computing machine seeking to compute an input's Dodgson score seems to require about logarithmically many adaptive calls to SAT.[3]

We call `GreedyScore` "greedy" because, as it sweeps through the votes, each swap it (virtually) does immediately improves the standing of the input candidate against some adversary that the input candidate is at that point losing to. The algorithm nonetheless is very simple. It limits itself to at most one swap per vote. Yet, its simplicity notwithstanding, we will eventually show that this (self-knowingly correct) algorithm is very frequently correct.

[3] We say "seems to," but we note that one can make a more rigorous claim here. As mentioned in Section 2, among the problems that Hemaspaandra, Hemaspaandra, and Rothe [1] prove complete for the language class Θ_2^p is `DodgsonWinner`. If one could, for example, compute Dodgson scores via a polynomial-time function-computing machine that made a (globally) constant-bounded number of queries to SAT, then this would prove that `DodgsonWinner` is in the boolean hierarchy [28], and thus that Θ_2^p equals the boolean hierarchy, which in turn would imply the collapse of the polynomial hierarchy [29]. That is, this function problem is so closely connected to a Θ_2^p-complete language problem that if one can save queries in the former, then one immediately has consequences for the complexity of the latter.

We now state the main result for this section, and a bit later we will briefly describe the algorithms in English.

Theorem 2. *1. `GreedyScore` (Algorithm 3.1) is self-knowingly correct for Score (recall that Score is defined in Section 2 in the statement of the `DodgsonScore` problem).*
2. `GreedyWinner` (Algorithm 3.2) is self-knowingly correct for `DodgsonWinner`.
3. `GreedyScore` and `GreedyWinner` both run in polynomial time.[4]

Note that Theorem 1.1 follows directly from Theorem 2.2. We will turn to Theorem 1.2 in Section 4.

Theorem 2, since it just states polynomial time, is not heavily dependent on the encoding scheme used. However, we will for specificity give a specific scheme that can be used. Note that the scheme we use will encode the inputs as binary strings by a scheme that is easy to compute and invert and encodes each vote as an $\mathscr{O}(\|C\| \log \|C\|)$-bit substring and each Dodgson triple as an $\mathscr{O}(\|V\| \cdot \|C\| \cdot \log \|C\|)$-bit string, where (C,V,c) is the input to the encoding scheme. For a Dodgson triple (C,V,c), our encoding scheme is as follows: First comes $\|C\|$, encoded as a binary string of length $\lceil \log(\|C\|+1) \rceil$,[5] preceded by the substring $1^{\lceil \log(\|C\|+1) \rceil}0$. Next comes the chosen candidate c, encoded as a binary string of length $\lceil \log(\|C\|+1) \rceil$. Finally each vote is encoded as a binary substring of length $\|C\| \cdot \lceil \log(\|C\|+1) \rceil$.

Regarding the notation used in Algorithm 3.1: A vote is represented as an array $v[]$ of length m, where $m = \|C\|$. For each vote $v[]$, $v[1]$ is the least preferred candidate, $v[2]$ is the second least preferred candidate, and so on, and $v[m]$ is the most preferred candidate. $Swap_i(v)$ means that the ith and $(i+1)$st values in $v[]$ are swapped.

We now describe in English what our algorithms actually do (and their detailed pseudocode—which is what is referred to by all references above and below to specific variables such as $v[]$, $Swap[]$, and $Deficit[]$—is also included). Briefly put, `GreedyScore`, for each candidate d, $c \neq d \in C$, computes (in $Deficit[d]$) the number of votes (if any) that c needs to gain in order to have strictly more votes than d (in a pairwise contest between them), and computes (in $Swaps[d]$) the number of votes v in which d is immediately adjacent to and preferred to c ($c \prec_v d$). If the former number is strictly greater than zero and the latter number is at least as large as the former number, then it

[4] The number of times lines of Algorithm 3.1 (respectively, Algorithm 3.2) are executed is clearly $\mathscr{O}(\|V\| \cdot \|C\|)$ (respectively, $\mathscr{O}(\|V\| \cdot \|C\|^2)$), and so these are indeed polynomial-time algorithms.

For completeness, we mention that when one takes into account the size of the objects being manipulated (in particular, under the assumption—which in light of the encoding scheme we will use below is not unreasonable—that it takes $\mathscr{O}(\log \|C\|)$ time to look up a key in either *Deficit* or *Votes* and $\mathscr{O}(\log \|V\|)$ time to update the associated value, and each *Swap* operation takes $\mathscr{O}(\log \|C\|)$ time) the running time of the algorithm might be more fairly viewed as $\mathscr{O}(\|V\| \cdot \|C\| \cdot (\log \|C\| + \log \|V\|))$ (respectively, $\mathscr{O}(\|V\| \cdot \|C\|^2 \cdot (\log \|C\| + \log \|V\|))$), though in any case it certainly is a polynomial-time algorithm.

[5] All logarithms in this paper are base 2. We use $\lceil \log(\|C\|+1) \rceil$-bit strings rather than $\lceil \log(\|C\|) \rceil$-bit strings as we wish to have the size of the coding scale at least linearly with the number of voters even in the pathological $\|C\| = 1$ case (in which each vote carries no information other than about the number of voters).

Algorithm 3.1 `GreedyScore`(C,V,c) [n = number of voters; m = number of candidates]

```
1: for all $d \in C-\{c\}$ do $Deficit[d] \leftarrow 1 - \lceil n/2 \rceil$; $Swaps[d] \leftarrow 0$ end for
2: for all votes $v[]$ in $V$ do
3:   $state \leftarrow$ "nocount"
4:   for all $i \in (1,\ldots,m)$ do
5:     if $(state =$ "incrdef"$) \vee (state =$ "swap"$)$ then
6:       $Deficit[v[i]] \leftarrow Deficit[v[i]] + 1$
7:       if $state =$ "swap" then $Swaps[v[i]] \leftarrow Swaps[v[i]] + 1$; $state \leftarrow$ "incrdef" end if
8:     else if $c = v[i]$ then
9:       $state \leftarrow$ "swap"
10:     end if
11:   end for
12: end for
13: $confidence \leftarrow$ "definitely"; $score \leftarrow 0$
14: for all $d \in C-\{c\}$ do
15:   if $Deficit[d] > 0$ then
16:     $score \leftarrow score + Deficit[d]$
17:     if $Deficit[d] > Swaps[d]$ then$confidence \leftarrow$ "maybe"; $score \leftarrow score + 1$ end if
18:   end if
19: end for
20: return $(score, confidence)$
```

Algorithm 3.2 `GreedyWinner`(C,V,c)

```
1: $(cscore, confidence) =$ GreedyScore$(C,V,c)$; $winner \leftarrow$ "yes"
2: for all candidates $d \in C-\{c\}$ do
3:   $(dscore, dcon) \leftarrow$ GreedyScore$(C,V,d)$
4:   if $dscore < cscore$ then $winner \leftarrow$ "no" end if
5:   if $dcon =$ "maybe" then $confidence \leftarrow$ "maybe" end if
6: end for
7: return $(winner, confidence)$
```

is the case that by adjacent swaps in exactly the former number of votes—when done in that number of votes chosen from among those votes v satisfying $c \prec_v d$—c can be with perfect efficiency (every swap pays off by reducing a positive shortfall) be changed to beating d. If the number values just stated are not the case, the `GreedyScore` algorithm declares that it is stumped by the current input. If it is stumped for no candidate d, $c \neq d \in C$, then it simply adds up the costs of defeating each other candidate, and is secure in the knowledge that this is optimal (see also the proof in the full version). Turning to the `GreedyWinner` algorithm, it does the above for all candidates, and if while doing so `GreedyScore` is never stumped, then `GreedyWinner` uses in the obvious way the information it has obtained, and (correctly) states whether c is a Dodgson winner of the input election.

Note that `GreedyWinner` could easily be modified into a new polynomial-time algorithm that, rather than checking whether a given candidate is the winner of the given

Dodgson election, finds all Dodgson winners by taking as input a Dodgson election alone (rather than a Dodgson triple) and outputting a list of *all* the Dodgson winners in the election. This modified algorithm on any Dodgson election (C,V) would make exactly the same calls to `GreedyScore` that the current `GreedyWinner` (on input (C,V,c), where $c \in C$) algorithm makes, and the new algorithm would be accurate whenever every call it makes to `GreedyScore` returns "definitely" as its second argument. Thus, whenever the current `GreedyWinner` would return a "definitely" answer so would the new Dodgson-winner-finding algorithm (when their inputs are related in the same manner as described above). These comments explain why in the title (and abstract), we were correct in speaking of "*finding* Dodgson-Election Winners" (rather than merely recognizing them).

4 Analysis of the Correctness Frequency of the Two Heuristic Algorithms

In this section, we show that, as long as the number of votes is much greater than the number of candidates, `GreedyWinner` is a frequently self-knowingly correct algorithm.

Theorem 3. *For each $m,n \in \mathbb{N}^+$, the following hold. Let $C = \{1,\dots,m\}$.*

1. *Let V satisfy $\|V\| = n$. For each $c \in C$, if for all $d \in C - \{c\}$ it holds that $\|\{i \in \{1,\dots,n\} \mid c <_{v_i} d\}\| \leq \frac{2mn+n}{4m}$ and $\|\{i \in \{1,\dots,n\} \mid c \prec_{v_i} d\}\| \geq \frac{3n}{4m}$ then* $\texttt{GreedyScore}(C,V,c) = (\mathit{Score}(C,V,c), \text{"definitely"})$.
2. *For each $c,d \in C$ such that $c \neq d$, $\Pr((\|\{i \in \{1,\dots,n\} \mid c <_{v_i} d\}\| > \frac{2mn+n}{4m}) \vee (\|\{i \in \{1,\dots,n\} \mid c \prec_{v_i} d\}\| < \frac{3n}{4m})) < 2e^{\frac{-n}{8m^2}}$, where the probability is taken over drawing uniformly at random an m-candidate, n-voter Dodgson election $V = (v_1,\dots,v_n)$ (i.e., all $(m!)^n$ Dodgson elections having m candidates and n voters have the same likelihood of being chosen).*
3. *For each $c \in C$, $\Pr(\texttt{GreedyScore}(C,V,c) \neq (\mathit{Score}(C,V,c), \text{"definitely"})) < 2(m-1)e^{\frac{-n}{8m^2}}$, where the probability is taken over drawing uniformly at random an m-candidate, n-voter Dodgson election $V = (v_1,\dots,v_n)$.*
4. $\Pr((\exists c \in C)[\texttt{GreedyWinner}(C,V,c) \neq (\texttt{DodgsonWinner}(C,V,c), \text{"definitely"})]) < 2(m^2-m)e^{\frac{-n}{8m^2}}$*, where the probability is taken over drawing uniformly at random an m-candidate, n-voter Dodgson election $V = (v_1,\dots,v_n)$.*

Note that Theorem 1.2 follows from Theorem 3.4.

The main intuition behind Theorem 3 is that, in any election having m candidates and n voters, and for any two candidates c and d, it holds that, in exactly half of the ways v a voter can vote, $c <_v d$, but for exactly $1/m$ of the ways, $c \prec_v d$. Thus, assuming that the votes are chosen independently of each other, when the number of voters is large compared to the number of candidates, with high likelihood the number of votes v for which $c <_v d$ will hover around $n/2$ and the number of votes for which $c \prec_v d$ will hover around n/m. This means that there will (most likely) be enough votes available for our greedy algorithms to succeed.

Throughout this section, we regard $V = (v_1, \ldots, v_n)$ as a sequence of n independent observations of a random variable γ whose distribution is uniform over the set of all votes over a set $C = \{1, 2, \ldots, m\}$ of m candidates, where γ can take, with equal likelihood, any of the $m!$ distinct total orderings over C. (This distribution should be contrasted with such work as that of, e.g., [30], which in a quite different context creates dependencies between voters' preferences.)

We now turn to Lemma 1, which is needed to support (the proof of) Theorem 3. Lemma 1 follows from the following variant of Chernoff's Theorem [31].

Theorem 4 ([32]). *Let $X_1, \ldots, X_n$ be a seq. of mutually indep. random variables. If there is a $p \in [0,1] \subseteq \mathbb{R}$ such that, for each $i \in \{1, \ldots, n\}$, $\Pr(X_i = 1 - p) = p$ and $\Pr(X_i = -p) = 1 - p$, then for all $a \in \mathbb{R}$ where $a > 0$ it holds that $\Pr(\Sigma_{i=1}^{n} X_i > a) < e^{-2a^2/n}$.*

Lemma 1. *1.* $\Pr(\|\{i \in \{1, \ldots, n\} \mid c <_{v_i} d\}\| > \frac{2mn+n}{4m}) < e^{\frac{-n}{8m^2}}$.
2. $\Pr(\|\{i \in \{1, \ldots, n\} \mid c \prec_{v_i} d\}\| < \frac{3n}{4m}) < e^{\frac{-n}{8m^2}}$.

We now have established (see also the here-omitted proofs, available in the TR version) Theorem 1: As mentioned in Section 3, Theorem 1.1 follows from Theorem 2.2. Theorem 1.2 follows from Theorem 3.4. We refer the reader to the full (TR) version of this paper for a discussion of various open directions, and a suggested broader framework for the study of frequently self-knowingly correct algorithms.

References

1. Hemaspaandra, E., Hemaspaandra, L., Rothe, J.: Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. Journal of the ACM **44**(6) (1997) 806–825
2. Bartholdi III, J., Tovey, C., Trick, M.: Voting schemes for which it can be difficult to tell who won the election. Social Choice and Welfare **6** (1989) 157–165
3. Condorcet, M.: Essai sur l'Application de L'Analyse à la Probabilité des Décisions Rendues à la Pluralité des Voix. (1785) Facsimile reprint of original published in Paris, 1972, by the Imprimerie Royale.
4. McLean, I., Urken, A.: Classics of Social Choice. University of Michigan Press, Ann Arbor, Michigan (1995)
5. Dodgson, C.: A method of taking votes on more than two issues, Clarendon Press, Oxford, pamphet (1876)
6. Black, D.: The Theory of Committees and Elections. Cambridge University Press (1958)
7. Nanson, E.: Methods of election. Transactions and Proceedings of the Royal Society of Victoria **19** (1882) 197–240
8. Borda, J.C.d.: Mémoire sur les élections au scrutin. Histoire de L'Académie Royale des Sciences Année 1781 (1784)
9. Papadimitriou, C., Zachos, S.: Two remarks on the power of counting. In: Proceedings of the 6th GI Conference on Theoretical Computer Science, Springer-Verlag *Lecture Notes in Computer Science #145* (1983) 269–276
10. Hemaspaandra, E., Spakowski, H., Vogel, J.: The complexity of Kemeny elections. Theoretical Computer Science **349**(3) (2005) 382–391
11. Rothe, J., Spakowski, H., Vogel, J.: Exact complexity of the winner problem for Young elections. Theory of Computing Systems **36**(4) (2003) 375–386

12. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. second edn. MIT Press/McGraw Hill (2001)
13. Ausiello, G., Crescenzi, P., Protasi, M.: Approximate solution of NP optimization problems. Theoretical Computer Science **150**(1) (1995) 1–55
14. Kaporis, A., Kirousis, L., Lalas, E.: The probabilistic analysis of a greedy satisfiability algorithm. In: Proceedings of the 10th Annual European Symposium on Algorithms, Springer-Verlag *Lecture Notes in Computer Science #2461* (2002) 574–585
15. Chang, L., Korsh, J.: Canonical coin changing and greedy solutions. Journal of the ACM **23**(3) (1976) 418–422
16. Protasi, M., Talamo, M.: A new probabilistic model for the study of algorithmic properties of random graph problems. In: Proceedings of the 4th Conference on Fundamentals of Computation Theory, Springer-Verlag *Lecture Notes in Computer Science #158* (1983) 360–367
17. Slavik, P.: A tight analysis of the greedy algorithm for set cover. Journal of Algorithms **25**(2) (1997) 237–254
18. Brown, D.: A probabilistic analysis of a greedy algorithm arising from computational biology. In: Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM Press (2001) 206–207
19. Goldberg, A.V., Marchetti-Spaccamela, A.: On finding the exact solution of a zero-one knapsack problem. In: Proceedings of the 16th ACM Symposium on Theory of Computing. (1984) 359–368
20. Downey, R., Fellows, M.: Parameterized complexity. Springer-Verlag (1999)
21. Levin, L.: Average case complete problems. SIAM Journal on Computing (1986)
22. Spielman, D., Teng, S.: Smoothed analysis: Why the simplex algorithm usually takes polynomial time. Journal of the ACM **51**(3) (2004) 385–463
23. Hemaspaandra, E., Hemaspaandra, L.: Computational politics: Electoral systems. In: Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science, Springer-Verlag *Lecture Notes in Computer Science #1893* (2000) 64–83
24. Spakowski, H., Vogel, J.: Θ_2^p-completeness: A classical approach for new results. In: Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer-Verlag *Lecture Notes in Computer Science #1974* (2000) 348–360
25. Spakowski, H., Vogel, J.: The complexity of Kemeny's voting system. In: Proceedings of the Workshop Argentino de Informática Teórica, Volume 30 of *Anales Jornadas Argentinas de Informática e Investigación Operativa*, SADIO (2001) 157–168
26. Hemaspaandra, E., Spakowski, H., Vogel, J.: The complexity of Kemeny elections. Technical Report Math/Inf/14/03, Institut für Informatik, Friedrich-Schiller-Universität, Jena, Germany (2003)
27. Wagner, K.: More complicated questions about maxima and minima, and some closures of NP. Theoretical Computer Science **51**(1–2) (1987) 53–80
28. Cai, J., Gundermann, T., Hartmanis, J., Hemachandra, L., Sewelson, V., Wagner, K., Wechsung, G.: The boolean hierarchy I: Structural properties. SIAM Journal on Computing **17**(6) (1988) 1232–1252
29. Kadin, J.: The polynomial time hierarchy collapses if the boolean hierarchy collapses. SIAM Journal on Computing **17**(6) (1988) 1263–1282 Erratum appears in the same journal, 20(2):404.
30. Raffaelli, G., Marsili, M.: Statistical mechanics model for the emergence of consensus. Physical Review E **72**(1) (2005) 016114
31. Chernoff, H.: A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. Annals of Mathematical Statistics **23** (1952) 493–509
32. Alon, N., Spencer, J.: The Probabilistic Method. second edn. Wiley–Interscience (2000)

Reductions for Monotone Boolean Circuits

Kazuo Iwama* and Hiroki Morizumi

Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{iwama, morizumi}@kuis.kyoto-u.ac.jp

Abstract. The large class, say *NLOG*, of Boolean functions, including 0-1 Sort and 0-1 Merge, have an upper bound of $O(n \log n)$ for their monotone circuit size, i.e., have circuits with $O(n \log n)$ AND/OR gates of fan-in two. Suppose that we can use, besides such normal AND/OR gates, any number of more powerful "F-gates" which realize a monotone Boolean function F with $r(\geq 2)$ inputs and $r'(\geq 1)$ outputs. Note that the cost of each AND/OR gate is one and we assume that the cost of each F-gate is r. Now we define: A Boolean function f in NLOG is said to be F-Easy if f can be constructed by a circuit with AND/OR/F gates whose total cost is $o(n \log n)$. In this paper we show that 0-1 Merge is not F-Easy for an arbitrary monotone function F such that $r' \leq r/\log r$.

1 Introduction

Suppose that we wish to construct a Boolean monotone circuit for 0-1 Merge by using ordinary AND and OR gates of fan-in two and (any number of) Majority gates of any number of inputs at any places. It is well known that, if we are allowed to use only AND/OR gates, then the circuit size must be $\Theta(n \log n)$. Here a size means the number of gates and each AND/OR gate has a unit cost. When we use a Majority gate of r inputs, we assume that such a gate has a cost of r (the reason is given later). It should be noted that a Majority gate of r inputs is realized by using $O(r \log r)$ AND/OR gates, and therefore, if its cost is $r \log r$, then Majority gates are obviously useless. Our setting of cost r is thus subtle, and our primary question in this paper is whether such Majority gates are substantially useful, or whether we can prove a circuit size of $o(n \log n)$ for 0-1 Merge by adding Majority gates. The motivation is as follows:

Our knowledge about the traditional (bounded-fan-in AND, OR and Negation gates are allowed) circuit complexity is summarized as follows: (i) If we cannot use Negation, i.e., for monotone circuits, there are exponential lower bounds (e.g., [16], [1], [7]). (ii) If we can use few Negations, i.e., at most $(1/6) \log \log n$ Negations, then there are also a superpolynomial lower bounds [3]. (iii)If we can use $\lceil \log(n+1) \rceil$ Negations, then all n inputs can be negated by using $O(n \log n)$ gates [5]. In other words, if we can obtain an $\omega(n \log n)$ lower bound for circuits with $\lceil \log(n+1) \rceil$ Negations, the same lower bound applies for general circuits.

* Supported in part by Scientific Research Grant, Ministry of Japan, 1609211 and 16300003.

The best lower bound for this type of circuits, however, is $(5+1/3)n + \log(n+1)/3-c$ for a two-output function (we can apply the lower bound on a circuit with $\lceil \log(n+1)\rceil - 1$ Negations for Parity in [18] to the lower bound on a circuit with $\lceil \log(n+1)\rceil$ Negations for (Parity, $\neg$Parity).) and $(7+1/3)n+\log(n+1)/3-c$ for an n-output function [18]. (iv) For general circuits (i.e., without any restriction for the number of Negations), the best lower bound is still $5n - o(n)$ [12] [8] in spite of the long history of research.

Looking at this series of facts, the challenging and somewhat realistic goal is to attain nonlinear lower bounds for circuits of type (iii), for example, for the circuit that negates all n inputs as mentioned above (which we call *Inverter*). Towards this goal, this paper proposes an approach based on reduction which is quite popular in many different contexts of complexity theory.

Our Contribution. In [5], Beals, Nishino and Tanaka studied Inverter. Reading this paper carefully it turns out that if we can use only $\lceil \log(n+1)\rceil$ Negations, then the way of using them is very restricted (this is the case not only for Inverter but also for many others including (Parity, $\neg$Parity)), i.e., such circuit includes *monotone* subcircuits which compute $\lceil \log(n+1)\rceil$ different threshold values with respect to the number of input one's, in particular, one of them must be a majority of n. This means, if we can prove some (e.g., nonlinear) lower bound for monotone Majority, then that bound also applies for Inverter. Thus proving lower bounds for Inverter can be reduced to proving (similar) lower bounds for *monotone* circuits, which appears to be much easier.

We further extend this reduction approach. Let NLOG be the class of (possibly multi-output) Boolean functions which have a monotone upper bound of $O(n \log n)$. Then f in NLOG is said to be *Maj-Easy* if f has a circuit C of size $o(n \log n)$ with AND/OR gates of fan-in two and Majority gates of any fan-in. (Recall that the cost of each AND/OR gate is one and the cost of each Majority gate of fan-in r is r.) Now one can easily see that a conventional (without Majority gates) nonlinear lower bound for monotone Majority (and therefore the same lower bound for Inverter, too) is attained if we can find a monotone Boolean function f such that f is Maj-Easy and f has a conventional monotone lower bound of $\Omega(n \log n)$.

Note that we already know that several Boolean functions do have a conventional monotone lower bound of $\Omega(n \log n)$, including Merge. So, we are done if we could prove that Merge is Maj-Easy, which is exactly the question raised at the beginning of this paper. Unfortunately, we can show that Merge is *not* Maj-Easy. In fact we can prove a stronger result, Merge is not F-Easy for any monotone Boolean function F of r inputs and up to $r/\log r$ outputs (under the similar definitions, see Sec. 2 for details). In other words, even if such an F-gate is arbitrarily powerful, it does not help to reduce the complexity of Merge if its cost is r and its output size is at most $r/\log r$.

Thus we have to seek other candidates. (Sort has also an $\Omega(n \log n)$ lower bound but it obviously does not help since Sort is more powerful than Merge.) One candidate we show in this paper is what we call an approximated Sort that is Maj-Easy (but is not known if it has an $\Omega(n \log n)$ lower bound).

Previous Work. Proving a superpolynomial lower bound for AND/OR/ Negation circuits that compute a particular NP language implies P$\neq$NP. However, research towards this ultimate goal has not been very successful. Schnorr first gave a nontrivial lower bounds of $2n$ using base B_2 [17]. After a number of improvements, Zwick gave a $4n$ lower bound for base U_2 [23]. Some ten years later, Lachish and Raz succeeded in improving this bound to $4.5n$ by using the new Strongly-Two-Dependent function [12]. Iwama and Morizumi [8] raised this to $5n$ by deeper analysis of [12], which is currently best. All those results are based on the so-called gate-elimination method, which many people believe has a clear limit of power for further improvement.

For monotone circuits, Razborov first proved a superpolynomial lower bound for a circuit computing Clique [16], which was later improved to an exponential lower bound by Andreev [1]. By combining their proof technique and other techniques, Amano and Maruoka obtained a superpolynomial lower bound for circuits which can use at most $(1/6)\log\log n$ Negation gates [3].

As mentioned before, Beals, Nishino and Tanaka studied the least amount of Negations to compute all Boolean functions. Especially they showed, for Inverter, a lower bound of $5n + 3\log(n+1) - c$ and an upper bound of $O(n \log n)$ which is conjectured optimal by them [5]. Their lower bound was improved to $(7 + 1/3)n + \log(n+1)/3 - c$ in [18]. [9] shows that for $\lceil \log(n+1) \rceil$-Negation circuits a $6n - \log(n+1) - c$ lower bound can be proved by using the (Parity, $\neg$Parity) function and an $8n - \log(n+1) - c$ lower bound can be proved by using Inverter. Interestingly, if the number of available Negations is less than $\lceil \log(n+1) \rceil$, even by one, the lower bound jumps. Shun and Tanaka proved an exponential lower bound for $(\lceil \log(n+1) \rceil - 1)$-Negation circuits. [19]

0-1 Sort, 0-1 Merge and Majority are all practically important and have a large literature for their circuit realization. Majority can be constructed by $O(n)$ AND/ OR/Negation gates (see e.g., Chap. 3.4 in [22]) and by $O(n \log n)$ AND/OR gates by using the famous sorting network [2]. (Valiant gave a completely different construction based on probabilistic method [21].) For its monotone lower bound, however, we have only linear ones. In 84, Dunne proved a $3.5n$ lower bound [6], and in 86, Long proved a $4n$ lower bound [11], but we did not have any further progress in the last two decades. By contrast, we have tight lower bounds for Sort and Merge. Lamagna and Savage [13] proved an $\Omega(n \log n)$ lower bound for Sort. Pippenger and Valiant [15] and Lamagna [10] proved independently $\Omega(n \log n)$ lower bound for Merge. Amano, Maruoka and Tarui [4] proved $\Theta(2^a n)$ for Merge in negation-limited circuits with $\log\log n - a$ Negations.

2 *F*-Easiness and Nonlinear Lower Bounds

In this paper, we mainly deal with the class, denoted by *NLOG*, of monotone Boolean functions which have circuits of $O(n \log n)$ AND/OR gates of fan-in two. One of such functions is MERGE(n, m) that is a collection of functions that merges two presorted binary sequence $x_1 \leq x_2 \leq \cdots \leq x_n$ and $x_{n+1} \leq x_{n+2} \leq \cdots \leq x_m$ into a sequence $y_1 \leq y_2 \leq \cdots \leq y_m$. In this paper we discuss only MERGE$(n, 2n)$.

Without otherwise stated, all circuits in this paper are also monotone. *A (monotone) circuit C* is a directed acyclic diagram consisting of *gates* and *links* as shown in Fig. 1. Each gate has *input* and *output terminals*. Each link connects an output terminal to an input terminal, where no two links cannot go to a single input terminal. Gates are associated with different types: An *AND* (similarly for *OR*) *gate* has two input and one output terminals, an *F-gate* has r input and r' output terminals and computes the Boolean function $F : \{0,1\}^r \rightarrow \{0,1\}^{r'}$. Different F-gates in the circuit may have different r/r' values (i.e., different gate sizes), but must compute the Boolean function of the same type. An *input gate* has one output and no input terminals and is labeled by a variable in $\{x_1, \dots, x_n\}$ or a constant 0 or 1, and an *output gate* has one input and no output terminals and is labeled by a variable in $\{y_1, \dots, y_m\}$.

Each gate has a cost. The cost of AND and OR gate is one, and that of an F-gate is always r regardless of the function F. An input and output gate has cost zero. The *size* of a circuit C is the sum of the costs of all gates in C. The size measure is also used for a Boolean function f: $size_F(f)$ is the minimum size of a circuit with AND/OR/F gates computing f. $size(f)$ is the minimum size of a circuit computing f in which only AND and OR gates can be used (without F-gates). Thus NLOG can be written as a family of Boolean functions f such that $size(f) = O(n \log n)$. Now we are ready to define F-Easy Boolean functions; intuitively F-Easy functions are those functions for which F-gates are substantially useful:

Definition 1. *Let f be in NLOG and F be a monotone function. f is said to be F-Easy if $size_F(f)$ is $o(n \log n)$. In particular, if F is Majority then f is called Maj-Easy.*

Now the next Theorem is immediate:

Theorem 1. *Suppose that a function f is Maj-Easy and $size(f)$ is $\Omega(n \log n)$. Then $size(Majority)$ is $\omega(n)$.*

Inverter is a Boolean function from $\{0,1\}^n$ into $\{0,1\}^n$ such that $y_i = \neg x_i$ for $1 \leq i \leq n$. A (not monotone) circuit C is said to be *FewNOT* if C uses at most $\lceil \log(n+1) \rceil$ Negation gates. $size_{FewNOT}(f)$ is the minimum number of AND/OR/Negation gates that are needed to realize f by a FewNOT circuit.

Theorem 2. *(Implicit in [5]).*

$$size(Majority) \leq size_{FewNOT}(Inverter)$$

Thus in order to prove a nonlinear lower bound for the FewNOT size of Inverter, it is enough to find a function f in NLOG such that f is Maj-Easy and $size(f) = \Omega(n \log n)$. Although details are omitted, Theorem 2 still holds if Majority is replaced by LogThreshold which is a threshold function having $\log n$ output gates $T^n_{n/2}$, $T^n_{n/4}$(or $T^n_{3n/4}$, controlled by extra input gates), $T^n_{n/8}$(or $T^n_{3n/8}$, $T^n_{5n/8}$, $T^n_{7n/8}$) and so on, and therefore Maj-Easy in the above sentence can also be replaced by LogThreshold-Easy.

3 Merge is Not F-Easy

Recall that $\Omega(n \log n)$ lower bounds for $size(f)$ are already known for some functions f in NLOG including Sort and Merge. Hence, by Theorem 1, our goal would be achieved if we could prove, for example, Merge is Maj-Easy. Unfortunately this is not the case (and neither for Sort since Sort operates exactly as Merge for the presorted inputs). In fact we can prove the following stronger result:

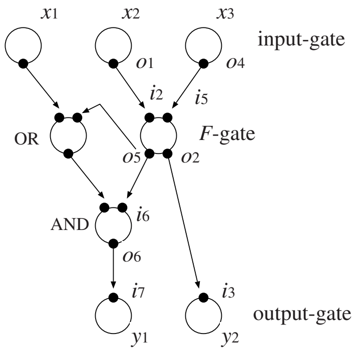


Fig. 1. circuit

Theorem 3. *For any monotone function $F : \{0,1\}^r \to \{0,1\}^{r'}$ such that $r' \le r/\log r$, MERGE$(n, 2n)$ is not F-Easy.*

To prove this theorem, we need several new definitions: We define vertex-disjoint paths for both circuits and graphs. For an directed acyclic graph $G = (V, E)$, a *path* is a sequence $v_1 v_2 \cdots v_k$ of vertices such that $(v_i, v_{i+1}) \in E$ for $1 \le i \le k-1$. Two paths $v_1 v_2 \cdots v_k$ and $u_1 u_2 \cdots u_l$ are said to be *vertex-disjoint* if $\{v_1, v_2, \ldots, v_k\} \cap \{u_1, u_2, \ldots, u_l\} = \emptyset$. For a circuit C, a path is a sequence of terminals $u_1 v_2 u_2 v_3 \cdots u_k v_{k+1}$ such that $u_1, u_2, \ldots, u_k$ are output terminals, $v_2, v_3, \ldots, v_{k+1}$ are input terminals, v_i and u_i $(2 \le i \le k)$ belong to the same gate, and there is an link from u_i to v_{i+1} $(1 \le i \le k)$. Two paths are *terminal-disjoint* if their terminals are disjoint. For example, $o_1 i_2 o_2 i_3$ and $o_4 i_5 o_5 i_6 o_6 i_7$ are terminal-disjoint in Fig. 1.

For finite sets X and Y $(|Y| \ge |X|)$, let δ be a one-to-one mapping from X into Y. We say that a graph $G = (V, E)$ *implements* a mapping $\delta : X \to Y$ if the following is met: (i) $X \subseteq V$ and $Y \subseteq V$. (ii) $X \cap Y = \emptyset$, and (iii) there are $|X|$ vertex-disjoint paths from each $x \in X$ to $\delta(x)$. For fixed X and Y, let M be a set of one-to-one mappings from X into Y. Then G *implements* M if G implements every mapping in M. Let $X = \{u_1, \ldots, u_n\}$ and $Y = \{w_1, \ldots, w_{2n}\}$. Then mapping $t_j (0 \le j \le n)$ is defined as $t_j(u_i) = w_{i+j}$. Let $T_n = \{t_0, t_1, \ldots, t_n\}$ (see Fig. 2). Then the following fact is known (see Corollary 2.2.2 in [15]).

Lemma 1. *Suppose that a graph $G = (V, E)$ implements T_n. Then $|E| = \Omega(n \log n)$.*

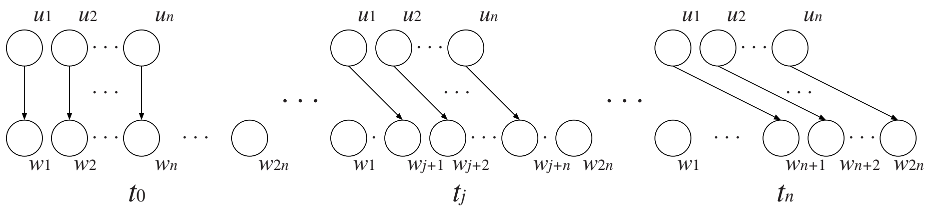


Fig. 2. T_n

Now we consider an arbitrary circuit, denoted by $C_{n,2n}$, with AND/OR/F gates which computes Merge$(n, 2n)$. Recall that an F-gate has r input and r' output terminals and different F-gates may have different r/r' values. Note that $C_{n,2n}$ has $2n$ input gates $x_1, \ldots, x_{2n}$ and $2n$ output gates $y_1, \ldots, y_{2n}$, and let mapping $t_j; \{x_1, \ldots, x_n\} \to \{y_1, \ldots, y_{2n}\}$ be defined exactly as before, i.e., $t_j(x_i) = y_{i+j}$.

Lemma 2. *For any $0 \leq j \leq n$, $C_{n,2n}$ has a set of n terminal-disjoint paths from x_i to $t_j(x_i)$.*

Proof. The following argument is similar to [15]. Suppose that $j = 0$. Then we set $x_1 = \cdots = x_n = 0$ and $x_{n+1} = \cdots = x_{2n} = 1$, which forces $y_1 = \cdots = y_n = 0$ and $y_{n+1} = \cdots = y_{2n} = 1$. Now we change the value of x_n from 0 to 1. Then since $C_{n,2n}$ is monotone, there must be at least one path P_0 from x_n to y_n where the value of all the (input and output) terminals on P_0 changes from 0 to 1 according to this input change. Note that these values of the terminals on P_0 will never change if we keep $x_n = x_{n+1} = \cdots = x_{2n} = 1$.

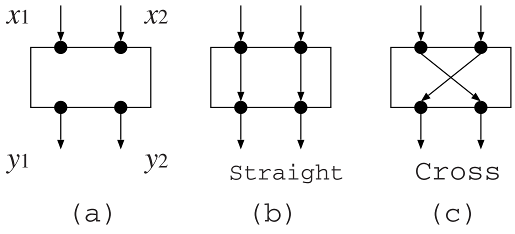


Fig. 3. switching gate

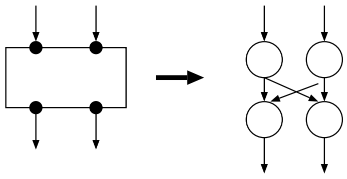


Fig. 4. replacement of switching gates

We next change the value of x_{n-1} from 0 to 1, by which the value of y_{n-1} changes from 0 to 1. Then there must be at least one new path P_1 from x_{n-1} to y_{n-1} such that all the terminal values on P_1 change from 0 to 1 and P_0 and P_1 are terminal-disjoint. (Otherwise, i.e., if all the new paths intersect with P_0, then the value of y_{n-1} should have been changed to 1 when x_n was changed from 0 to 1 in the previous step.) Similarly, by changing the value of x_{n-2} from 0 to 1, we can create another new path P_2 which does not intersect P_0 or P_1, and so on. Thus there are terminal-disjoint paths $P_0, \ldots, P_n$ from x_n to y_n $(= t_0(x_n))$, …, x_1 to y_1 $(= t_0(x_1))$, respectively.

For $j = 1$, we can repeat the same argument by initially setting $x_1 = \cdots = x_{n+1} = 0$ and $x_{n+2} = \cdots = x_{2n} = 1$. Similarly for $j = 2, 3, \ldots, n$. □

Lemma 3. *Suppose that every F-gate of r input and r' output terminals in $C_{n,2n}$ satisfies that $r' \leq r/\log r$ and that the size of $C_{n,2n}$ is s. Then there exists a graph $G_{n,2n} = (V, E)$ such that $G_{n,2n}$ implements T_n and $|E| = c \cdot s$ for some constant c.*

Proof. We consider a so-called *permutation network*, denoted by $\Pi_{r,r'}$, from r inputs to r' outputs in which we can use only *switching gates*. (In this paper we assume $r' \leq r$.) As shown in Fig. 3 (a), a switching gate has two input terminals x_1 and x_2, two output terminals y_1 and y_2 and two states called *Straight* and *Cross*. As shown in Fig. 3 (b) ((c),resp.) if the state is Straight (Cross, resp.) (y_1, y_2) is connected to (x_1, x_2) $((x_2, x_1)$, resp.). $\Pi_{r,r'}$ must realize every permutation between any size-r' subset of the inputs and the set of outputs by setting the state of each switching gate appropriately. By the method of [14], we can construct $\Pi_{r,r'}$ by using $O(r + r' \log r')$ switching gates.

Now we construct the graph $G_{n,2n}$ from the circuit $C_{n,2n}$ as follows. Each AND/OR gate of $C_{n,2n}$ is replaced by a vertex of in-degree two. Each F-gate is replaced by $\Pi_{r,r'}$ and then each switching gate in $\Pi_{r,r'}$ is replaced by the graph with four vertices as shown in Fig. 4, which creates $O(r)$ edges since $r' \leq r/\log r$. Therefore the number of edges in the whole resulting graph $G_{n,2n}$ is at most $c \cdot s$ for some constant c since the size of the original $C_{n,2n}$ is s. By Lemma 2, the original $C_{n,2n}$ has n terminal-disjoint paths from x_i to $t_j(x_i)$, which define a one-to-one mapping for each F-gate, from some subset of its input terminals to its output terminals. This mapping is realized by some permutation of the replaced $\Pi_{r,r'}$ and it is not hard to see that the permutation defines a set of vertex-disjoint paths in the graph obtained by the replacement of Fig. 4. Thus the original terminal-disjoint paths are transformed into n vertex-disjoint paths in $G_{n,2n}$, namely, it implements T_n. □

Proof of Theorem 3. Suppose for contradiction that MERGE$(n, 2n)$ is F-Easy. By definition, there is a circuit $C_{n,2n}$ which satisfies the condition of Lemma 3 and whose size is $o(n \log n)$. Then Lemma 3 implies that there is a graph $G = (V, E)$ which implements T_n and $|E| = o(n \log n)$, contradicting to Lemma 1. □

4 Concluding Remarks

Thus we need to seek another function towards our goal. One such candidate is what we call *d-Approximated Sort* defined as follows:

Definition 2. *Let $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ be a Boolean function and let m be the number of input one's and m' be output one's. Then f is called d-Approximated Sort if $y_1 \leq y_2 \leq \ldots \leq y_n$ and $|m - m'| \leq n/d$.*

$o(\log n)$-Approximated Sort is Maj-Easy for the following reason: (i) We can compute the n-input threshold function of any specific threshold value by using $2n$-input Majority by setting $\{0,1\}$'s to the extra n inputs appropriately. (ii) Let $d = o(\log n)$. Then d-Approximated Sort can be constructed by using d n-input threshold functions whose d threshold values distribute evenly between 0 and n. At this moment, we do not have any nontrivial lower bounds for its size.

Acknowledgments. The first author would like to thank Mike Paterson for valuable comments.

References

1. A. E. Andreev. On a Method for Obtaining Lower Bounds for the Complexity of Individual Monotone Functions. *Sov. Math. Doklady* 31(3), pp. 530–534, 1985.
2. M. Ajtai, J. Komlós and E. Szemerédi. An $O(n \log n)$ Sorting Network. *Proc. 15th STOC*, pp. 1–9, 1983.
3. K. Amano and A. Maruoka. A Superpolynomial Lower Bound for a Circuit Computing the Clique Function with at most $(1/6) \log \log n$ Negation Gates. *SIAM J. Comput.* 35(1), pp. 201–216, 2005.
4. K. Amano, A. Maruoka and J. Tarui. On the Negation-Limited Circuit Complexity of Merging. *Discrete Applied Mathematics* 126(1), pp. 3–8, 2003.
5. R. Beals, T. Nishino and K. Tanaka. On the Complexity of Negation-Limited Boolean Networks. *SIAM J. Comput.* 27(5), pp. 1334–1347, 1998.
6. P. E. Dunne. Lower Bound on the Monotone Network Complexity of Threshold Functions. *Proc. 22nd Ann. Allerton Conf. on Communication, Control and Computing*, pp. 911–920, 1984.
7. D. Harnik and R. Raz. Higher Lower Bounds on Monotone Size. *Proc. 32th STOC*, pp. 378–387, 2000.
8. K. Iwama and H. Morizumi. An Explicit Lower Bound of $5n - o(n)$ for Boolean Circuits. *Proc. 27th MFCS*, pp. 353–364, 2002.
9. K. Iwama, H. Morizumi and J. Tarui. Lower Bounds on the Negation-Limited Circuit Complexity. *manuscript*, 2006.
10. E. A. Lamagna. The Complexity of Monotone Networks for Certain Bilinear Forms, Routing Problems, Sorting, and Merging. *IEEE Trans. Computers* 28(10), pp. 773–782, 1979.
11. D. Long. The Monotone Circuit Complexity of Threshold Functions. Unpublished manuscript, University of Oxford, 1986.
12. O. Lachish and R. Raz. Explicit Lower Bound of $4.5n - o(n)$ for Boolean Circuits. *Proc. 33th STOC*, pp. 399–408, 2001.

13. E. A. Lamagna and J. E. Savage. Combinational Complexity of Some Monotone Functions. *Proc. 15th Ann. IEEE Symp. on Switching and Automata Theory*, pp. 140–144, 1974.
14. A. Y. Oruç. A Study of Permutation Networks: New Designs and Some Generalizations. *J. Parallel Distrib. Comput.* 22(2), pp. 359–366, 1994.
15. N. Pippenger and L. G. Valiant. Shifting Graphs and Their Applications. *J. ACM* 23(3), pp. 423–432, 1976.
16. A. A. Razborov. Lower Bounds on the Monotone Complexity of Some Boolean Functions. *Sov. Math. Doklady* 31, pp. 354–357, 1985.
17. C. Schnorr. Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. *Computing* 13, pp. 155–171, 1974.
18. S. C. Sung and K. Tanaka. Lower Bounds on Negation-Limited Inverters. *Proc. 2nd DMTCS*, pp. 360–368, 1999.
19. S. C. Sung and K. Tanaka. An Exponential Gap with the Removal of One Negation Gate. *Inf. Process. Lett.* 82(3), pp. 155–157, 2002.
20. K. Tanaka, T. Nishino and R. Beals. Negation-Limited Circuit Complexity of Symmetric Functions. *Inf. Process. Lett.* 59(5), pp. 273–279, 1996.
21. L. G. Valiant. Short Monotone Formulae for the Majority Function. *J. Algorithms* 5(3), pp. 363–366, 1984.
22. I. Wegener. The Complexity of Boolean Functions. *Wiley-Teubner Series in Computer Science*, 1987.
23. U. Zwick. A 4n Lower Bound on the Combinatorial Complexity of Certain Symmetric Boolean Functions over the Basis of Unate Dyadic Boolean Functions. *SIAM J. Comput.* 20, pp. 499–505, 1991.

Generalised Integer Programming Based on Logically Defined Relations

Peter Jonsson[⋆] and Gustav Nordh[⋆⋆]

Department of Computer and Information Science,
Linköpings Universitet S-581 83 Linköping, Sweden
{petej, gusno}@ida.liu.se

Abstract. Many combinatorial optimisation problems can be modelled as integer linear programs. We consider a class of generalised integer programs where the constraints are allowed to be taken from a broader set of relations (instead of just being linear inequalities). The set of allowed relations is defined using a many-valued logic and the resulting class of relations have provably strong modelling properties. We give sufficient conditions for when such problems are polynomial-time solvable and we prove that they are **APX**-hard otherwise.

1 Introduction

Combinatorial optimisation problems can often be formulated as *integer programs*. In its most general form, the aim in such a program is to assign integers to a set of variables such that a set of linear inequalities are satisfied and a linear goal function is maximised. That is, one wants to solve the optimisation problem

$$\begin{aligned} &\max c^T x \\ &Ax \geq b \\ &x \in \mathbb{N}^n \end{aligned}$$

where A is an $m \times n$ rational matrix, b is a rational m-vector, and c is a rational n-vector. It is widely acknowledged that many real-world problems can be conveniently captured by integer programs. In its general form, integer programming is **NP**-complete to solve exactly [4].

Many restricted variants of the integer programming problem are still fairly expressive and have gained much attention in the literature. One such restriction is to restrict x such that $x \in \{0, 1\}^n$. This problem, commonly called MAXIMUM 0-1 PROGRAMMING, is still very hard, in fact it is **NPO**-complete [9]. Another common restriction is to only allow c to contain non-negative values. Under these restrictions, the complexity and approximability of MAXIMUM 0-1 PROGRAMMING is very well-studied, cf. [3,10]. In fact, much is known even if the constraints imposed by the inequality $Ax \geq b$ are replaced by other types of constraints. For instance, MAX ONES is MAXIMUM 0-1 PROGRAMMING with non-negative weights and arbitrary constraints over

[⋆] Partially supported by the *Center for Industrial Information Technology* (CENIIT) under grant 04.01, and by the *Swedish Research Council* (VR) under grant 621-2003-3421.

[⋆⋆] Supported by the *National Graduate School in Computer Science* (CUGS), Sweden.

$\{0,1\}$. The approximability of Max Ones has been completely classified for every set Γ of allowed constraints over $\{0,1\}$ [10].

In this paper, we consider a class of generalised integer programming problems where the variable domains are finite (but not restricted to be 2-valued); and the constraints are allowed to be taken from a set of logically defined relations. The set of relations that we consider is based on *regular signed logic* [6], where the underlying finite domain is a totally-ordered set of integers $\{0,\ldots,d\}$. This logic provides us with convenient concepts for defining a class of relations with strong modelling capabilities. Jeavons and Cooper [8] have proved that any constraint can be expressed as the conjunction of expressions over this class of relations. A disadvantage with their approach is that the resulting set of constraints may by exponentially large (in the number of tuples in the constraint to be expressed). An improved algorithm solving the same problem has been given by Gil et al. [5]. It takes a constraint/relation represented by the set of all assignments/tuples that satisfies it and outputs in polynomial time (in the number of tuples) an expression that is equivalent to the original constraint.

The complexity of reasoning within this class of logically defined relations has been considered before in, for example, [2,8]. However, optimisation within this framework has not been considered earlier. To make the presentation simpler, let Max Sol denote the maximisation problem restricted to positively weighted objective functions, Min Sol denote the minimisation version of Max Sol, and let Max AW Sol denote the problem without restrictions on the weights (here, AW stands for arbitrary weights). The reader is referred to Section 2 for exact definitions.

Let $\mathcal{R}$ denote the class of relations that can be defined by the regular signed logic. Our aim is to, given a subset Γ of $\mathcal{R}$, classify the complexity of the optimisation problem when the constraints are restricted to relations in Γ. Thus, we parameterise our problems according to the allowed relations and we denote the restricted problems Max Sol(Γ), Min Sol(Γ), and Max AW Sol(Γ).

Our main results are: For these three problems, we give sufficient conditions for when they are polynomial-time solvable and we prove that they are **APX**-hard otherwise. We also show that the tractable fragments can be efficiently recognised. When a problem is **APX**-hard, then there is a constant c such that the problems cannot be approximated in polynomial time within $c-\varepsilon$ for any $\varepsilon > 0$ unless **P**=**NP**. A direct consequence is that these problems do not admit polynomial-time approximation schemes. This kind of dichotomy results are important in computational complexity since they can be seen as exceptions to Ladner's [11] result; he proved that there exists an infinite hierarchy of increasingly difficult problems between **P** and the **NP**-complete problems. Thus, the existence of a complexity dichotomy for a class of problems cannot be taken for granted.

There has been much research on combining integer (linear) programming and logic/constraint programming in order to benefit from strengths of the respective areas (cf. [7,14]). One approach is to increase the modeling power of integer (linear) programming by allowing models to contain logic formulas. Our work can be seen as a crude estimation for the price, in terms of computational complexity, that comes with the additional expressive power of such an approach. Our results can also be seen as a first step towards extending the approximability classification for MAX ONES in [10] to non-Boolean domains. One of the observations of [10] was that none of the (infinite number) of combinatorial optimisation problems captured by MAX ONES have a polynomial time approximation scheme (unless **P**=**NP**). Our results strongly indicates that the situation remains the same for MAX ONES generalised to arbitrary finite domains. Our work also complement the recent dichotomy result due to Creignou et al. [2] for the decision version (decide whether there is a solution at all) of exactly the same framework that we study in this paper.

Our results are to a certain extent based on recent algebraic methods for studying *constraint satisfaction problems*. The use of algebraic techniques for studying such problems has made it possible to clarify the borderline between polynomial-time solvable and intractable cases. Both our tractability and hardness results exploit algebraic techniques – typically, we prove a restricted base case and then extend the result to its full generality via algebraic techniques. To this end, we introduce the concept of *max-cores* (which is a variant of the algebraic and graph-theoretic concept *core*).

The paper is structured as follows: Section 2 contains some basic definitions of constraint satisfaction and logical methods for constructing constraint languages. Section 3 presents the methods used for proving the results and the main results for the three problems are presented in Sections 4 and 5. Finally, some concluding remarks are given in Section 6. Due to space limitations, many proofs have been omitted[1].

2 Integer Programming and Logic

We begin by presenting the *constraint satisfaction problem* and how it is connected with integer programming. We continue by showing how logics can be used for defining relations suited for integer programming.

We define constraint satisfaction as follows: Let the *domain* $D = \{0, 1, \ldots, d\}$ be equipped with the total order $0 < 1 < \ldots < d$. The set of all n-tuples of elements from D is denoted by D^n. Any subset of D^n is called an n-ary relation on D. The set of all finitary relations over D is denoted by R_D. A constraint language over D is a finite set $\Gamma \subseteq R_D$. Given a relation R, we let $ar(R)$ denote the arity of R. Constraint languages are the way in which we specify restrictions on our problems. The constraint satisfaction problem over the constraint language Γ, denoted $\text{CSP}(\Gamma)$, is defined to be the decision problem with instance (V, D, C), where V is a set of variables, D is a domain, and C is a set of constraints $\{C_1, \ldots, C_q\}$, in which each constraint C_i is a pair (ϱ_i, s_i) with s_i a list of variables of length m_i, called the constraint scope, and ϱ_i an m_i-ary relation over the set D, belonging to Γ, called the constraint relation.

The question is whether there exists a solution to (V, D, C) or not, that is, a function from V to D such that, for each constraint in C, the image of the constraint scope is a member of the constraint relation. The optimisation problem we are going to study, MAX SOL, can then be defined as follows:

Definition 1. *Weighted Maximum Solution* over the constraint language Γ, denoted $\text{MAX SOL}(\Gamma)$, is defined to be the optimisation problem with

[1] A full version of the paper can be found at: `www.ida.liu.se/~gusno/MFCS06L.pdf`

Instance: Tuple (V, D, C, w), where (V, D, C) is a CSP instance over Γ, and $w : V \to \mathbb{N}$ is a weight function.

Solution: An assignment $f : V \to D$ to the variables such that all constraints are satisfied.

Measure: $\sum_{v \in V} w(v) \cdot f(v)$

We will also consider the analogous minimisation problem MIN SOL and the problem MAX AW SOL where the weights are arbitrary, i.e., not assumed to be non-negative. By choosing Γ appropriately, many integer programming problems can be modelled by using these problems. For instance, the well-known problem INDEPENDENT SET (where the objective is to find an independent set of maximum weight in an undirected graph) can be viewed as the MAX SOL($\{(0,0),(0,1),(1,0)\}$ problem. Sometimes we will consider problems restricted to instances where each variable may occur at most k times, denoted MAX SOL(Γ)-k. Next, we consider a framework based on regular signed logic for expressing constraint languages using logic which was introduced by Creignou et al. in [2].

Let V be a set of variables. For $x \in V$ and $a \in D$, the inequalities $x \geq a$ and $x \leq a$ are called positive and negative literals, respectively. A *clause* is a disjunction of literals. A *clausal pattern* is a multiset of the form $P = (+a_1, \ldots, +a_p, -b_1, \ldots, -b_q)$ where $p, q \in \mathbb{N}$ and $a_i, b_i \in D$ for all i. The pattern P is said to be *negative* if $p = 0$ and *positive* if $q = 0$. The sum $p + q$, also denoted $|P|$, is the *length* of the pattern.

A *clausal language* L is a set of clausal patterns. Given a clausal language L, an L-*clause* is a pair $(P, \mathbf{x})$, where $P \in L$ is a pattern and $\mathbf{x}$ is a vector of not necessarily distinct variables from V such that $|P| = |\mathbf{x}|$. A pair $(P, \mathbf{x})$ with a pattern $P = (+a_1, \ldots, +a_p, -b_1, \ldots, -b_q)$ and variables $\mathbf{x} = (x_1, \ldots, x_{p+q})$ represents the clause $(x_1 \geq a_1 \vee \ldots \vee x_p \geq a_p \vee x_{p+1} \leq b_1 \vee \ldots \vee x_{p+q} \leq b_q)$, where $\vee$ is the disjunction operator. An L-formula φ is a conjunction of a finite number of L-clauses. An *assignment* is a mapping $I : V \to D$ assigning a domain element $I(x)$ to each variable $x \in V$ and I *satisfies* φ if and only if $(I(x_1) \geq a_1 \vee \ldots \vee I(x_p) \geq a_p \vee I(x_{p+1}) \leq b_1 \vee \ldots \vee I(x_{p+q}) \leq b_q)$ holds for every clause in φ. It can be easily seen that the literals $+0$ and $-d$ are superfluous since the inequalities $x \geq 0$ and $x \leq d$ vacuously hold. Without loss of generality, it is sufficient to only consider patterns and clausal languages without such literals. We see that clausal patterns are nothing more than a convenient way of specifying certain relations – consequently, we can also use them for defining constraint languages. Thus, we make the following definitions: Given a clausal language L and a clausal pattern $P = (+a_1, \ldots, +a_p, -b_1, \ldots, -b_q)$, we let $Rel(P)$ denote the corresponding relation, i.e., $Rel(P) = \{\mathbf{x} \in D^{p+q} \mid (P, \mathbf{x}) \text{ hold}\}$ and $\Gamma_L = \{Rel(P) \mid P \in L\}$.

It is easy to see that several well studied optimisation problems are captured by this framework.

Example 2. Let the domain D be $\{0, 1\}$, then MAX SOL($(-0, -0)$) is exactly MAXIMUM INDEPENDENT SET, and MIN SOL($(+1, +1)$) is exactly MINIMUM VERTEX COVER.

3 Methods

3.1 Approximability and Reductions

A *combinatorial optimisation problem* is defined over a set $\mathcal{I}$ of *instances* (admissible input data); each instance $I \in \mathcal{I}$ has a finite set $\mathsf{sol}(I)$ of *feasible solutions* associated with it. The objective is, given an instance I, to find a feasible solution of *optimum* value with respect to some measure function $m : \mathcal{I} \times \mathsf{sol}(I) \to \mathbb{N}$. The optimal value is the largest one for *maximisation* problems and the smallest one for *minimisation* problems. A combinatorial optimisation problem is in **NPO** if its instances and solutions can be recognised in polynomial time, the solutions are polynomially-bounded in the input size, and the measure function can be computed in polynomial time (see, e.g., [1]).

We say that a solution $s \in \mathsf{sol}(I)$ to an instance I of an **NPO** problem Π is r-approximate if it is satisfying $\max\left\{\frac{m(I,s)}{\textsc{opt}(I)}, \frac{\textsc{opt}(I)}{m(I,s)}\right\} \leq r$, where $\textsc{opt}(I)$ is the optimal value for a solution to I. An approximation algorithm for an **NPO** problem Π has *performance ratio* $\mathcal{R}(n)$ if, given any instance I of Π with $|I| = n$, it outputs an $\mathcal{R}(n)$-approximate solution.

Let **PO** denote the class of **NPO** problems that can be solved (to optimality) in polynomial time. An **NPO** problem Π is in the class **APX** if there is a polynomial-time approximation algorithm for Π whose performance ratio is bounded by a constant. Completeness in **APX** is defined using a reduction called AP-reduction [3,10]. An **NPO** problem Π is **APX**-*hard* if every problem in **APX** is AP-reducible to it. If, in addition, Π is in **APX**, then Π is called **APX**-*complete*. It is well-known (and not difficult to prove) that every **APX**-hard problem is **NP**-hard. In our proofs it will be more convenient for us to use another type of approximation-preserving reduction called L-reduction [1].

Definition 3. An **NPO** problem Π_1 is said to be *L-reducible* to an **NPO** problem Π_2 if two polynomial-time computable functions F and G and positive constants β and γ exist such that

(a) given any instance I of Π_1, algorithm F produces an instance $I' = F(I)$ of Π_2, such that the measure of an optimal solution for I', $\textsc{opt}(I')$, is at most $\beta \cdot \textsc{opt}(I)$;
(b) given $I' = F(I)$, and any solution s' to I', algorithm G produces a solution s to I such that $|m_1(I, s) - \textsc{opt}(I)| \leq \gamma \cdot |m_2(I', s') - \textsc{opt}(I')|$, where m_1 is the measure function for Π_1 and m_2 is the measure function for Π_2.

A well-known fact (see, e.g., Lemma 8.2 in [1]) is that if Π_1 is L-reducible to Π_2 and $\Pi_1 \in$ **APX** then there is an AP-reduction from Π_1 to Π_2. Hence, when proving **APX**-hardness results we can use L-reductions instead of AP-reductions as long as the problem we are reducing from is in **APX**. It is well-known (cf. [1, Corr. 3.13]) that if **P** $\neq$ **NP**, then no **APX**-complete problem can have a **PTAS**.

3.2 Algebraic Framework

An operation on D is an arbitrary function $f : D^k \to D$. Any operation on D can be extended in a standard way to an operation on tuples over D, as follows: Let f be a k-ary operation on D and let R be an n-ary relation over D. For any collection of k tuples,

$\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R$, the n-tuple $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k})$ is defined as follows: $f(\boldsymbol{t_1}, \ldots, \boldsymbol{t_k}) = (f(\boldsymbol{t_1}[1], \ldots, \boldsymbol{t_k}[1]), \ldots, f(\boldsymbol{t_1}[n], \ldots, \boldsymbol{t_k}[n]))$ where $\boldsymbol{t_j}[i]$ is the i-th component in tuple $\boldsymbol{t_j}$. If f is an operation such that for all $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R$ $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}) \in R$, then R is said to be invariant under f. If all constraint relations in Γ are invariant under f then Γ is invariant under f. An operation f such that Γ is invariant under f is called a polymorphism of Γ. The set of all polymorphisms of Γ is denoted $Pol(\Gamma)$. Given a set of operations F, the set of all relations that is invariant under all the operations in F is denoted $Inv(F)$. Sets of operations of the form Pol(Γ) are known as *clones*, and they are well-studied objects in algebra (cf. [12]).

A first-order formula φ over a constraint language Γ is said to be *primitive positive* (or pp-formula for short) if it is of the form $\exists \mathbf{x} : (R_1(\mathbf{x}_1) \wedge \ldots \wedge R_k(\mathbf{x}_k))$ where $R_1, \ldots, R_k \in \Gamma$ and $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are vectors of variables such that $ar(R_i) = |\mathbf{x}_i|$ for all i. Note that a pp-formula φ with m free variables defines an m-ary relation $R \subseteq D^m$, denoted $R \equiv_{pp} \varphi$; the relation R is the set of all m-tuples satisfying the formula φ.

We define a closure operation $\langle \cdot \rangle$ such that $R \in \langle \Gamma \rangle$ if and only if the relation R can be obtained from Γ by pp-formulas. The following lemma states that MAX SOL over finite subsets of $\langle \Gamma \rangle$ is no harder than MAX SOL over Γ itself. The lemma provides a strong approximation preserving reduction (sometimes called S-reduction) which is simultaneously an AP- and L-reduction and preserves membership in approximations classes such as **PO** and **APX**.

Lemma 4. *Let Γ and Γ' be finite constraint languages such that $\Gamma' \subseteq \langle \Gamma \rangle$. If* MAX SOL$(\Gamma)$ *is in* **PO**, *then* MAX SOL(Γ') *is in* **PO** *and if* MAX SOL(Γ') *is* **APX**-*hard then* MAX SOL(Γ) *is* **APX**-*hard.*

Proof. We give an approximation preserving reduction from MAX SOL(Γ') to MAX SOL(Γ). Consider an instance $I = (V, D, C, w)$ of MAX SOL(Γ'). We transform I into an instance $F(I) = (V', D, C', w')$ of MAX SOL(Γ). For every constraint $C = (R, (v_1, \ldots, v_m))$ in I, R can be represented as

$$\exists_{v_{m+1}}, \ldots, \exists_{v_n} : R_1(v_{11}, \ldots, v_{1n_1}) \wedge \cdots \wedge R_k(v_{k1}, \ldots, v_{kn_k})$$

where $R_1, \ldots, R_k \in \Gamma \cup \{=_D\}$, $v_{m+1}, \ldots, v_n$ are fresh variables, and $v_{11}, \ldots, v_{1n_1}, v_{21}, \ldots, v_{kn_k} \in \{v_1, \ldots, v_n\}$. Replace the constraint C with the constraints $(R_1, (v_{11}, \ldots, v_{1n_1})), \ldots, (R_k, (v_{k1}, \ldots, v_{kn_k}))$, add $v_{m+1}, \ldots, v_n$ to V, and extend w so that $v_{m+1}, \ldots v_n$ are given weight 0. If we repeat the same reduction for every constraint in C it results in an equivalent instance of MAX SOL$(\Gamma \cup \{=_D\})$.

Each equality constraint can be removed by identifying variables that are forced to be equal, replacing them with a single variable, and updating the weight function w accordingly. The resulting instance $F(I) = (V', D, C', w')$ of MAX SOL(Γ) has the same optimum as I (i.e., $\text{OPT}(I) = \text{OPT}(F(I))$) and can be obtained in polynomial time. Now, given a feasible solution s' for $F(I)$, let $G(I, s')$ be the feasible solution for I where: The variables in I assigned by s' inherit their value from s'; the variables in I which are still unassigned all occur in equality constraints and their values can be found by simply propagating the values of the variables which have already been assigned. It should be clear that $m(I, G(I, s')) = m(F(I), s')$ for any feasible solution s' for $F(I)$. Hence, the functions F and G, as described above, is an approximation preserving reduction from MAX SOL(Γ') to MAX SOL(Γ). □

The lemma above obviously holds also for MIN SOL and MAX AW SOL. Also note the following consequence: if Γ and Γ' are finite constraint languages such that $\langle \Gamma' \rangle = \langle \Gamma \rangle$, then MAX SOL$(\Gamma)$ is **APX**-hard (in **PO**) if and only if MAX SOL(Γ') is **APX**-hard (in **PO**).

The next lemma simplifies some of the forthcoming proofs and its proof is easy.

Lemma 5. *Let* $P = (+a_1, +a_2, \ldots, +a_p, -b_1, \ldots, -b_q)$ *and* $P_1 = (+a_1, +\min\{a_2, \ldots, a_p\}, -b_1, \ldots, -b_q)$. *Then,* **APX**-*hardness of* MAX SOL(Γ_{P_1}) *implies the* **APX**-*hardness of* MAX SOL(Γ_P). *Similarly, if* $P_2 = (+a_1, \ldots, +a_p, -b_1, -\max\{b_2, \ldots, b_q\})$, *then* **APX**-*hardness of* MAX SOL(Γ_{P_2}) *implies* **APX**-*hardness of* MAX SOL(Γ_P). *The same results also holds for* MIN SOL *and* MAX AW SOL.

For a relation $R = \{(d_{11}, \ldots, d_{1m}), \ldots, (d_{t1}, \ldots, d_{tm})\}$ and a unary operation f, let $f(R)$ denote the relation $\{(f(d_{11}), \ldots, f(d_{1m})), \ldots, (f(d_{t1}), \ldots, f(d_{tm}))\}$. Similarly, let $f(\Gamma)$ denote the constraint language $\{f(R) \mid R \in \Gamma\}$.

Lemma 6. *Let* Γ *be a finite constraint language over* D *and* f *a unary operation in* $Pol(\Gamma)$ *such that* $f(d) \geq d$ *for all* $d \in D$. *If* MAX SOL$(f(\Gamma))$ *is* **APX**-*complete, then* MAX SOL(Γ) *is* **APX**-*hard, and if* MAX SOL$(f(\Gamma))$ *is in* **PO**, *then so is* MAX SOL(Γ).

Proof. We prove that MAX SOL$(f(\Gamma))$ is L-reducible to MAX SOL(Γ). Given an instance $I = (V, D, C, w)$ of MAX SOL$(f(\Gamma))$ we let $F(I) = (V, D', C', w)$ be the instance of MAX SOL(Γ) where every constraint relation $f(R_i)$ occurring in a constraint $C_i \in C$ has been replaced by R_i. Given a solution s' of $F(I)$, let $G(I, s')$ be the solution s of I where $s(x) = f(s'(x))$ for each variable x. Since $f \in Pol(\Gamma)$ and $f(d) \geq d$ for all $d \in D$, we have that $\text{OPT}(I) = \text{OPT}(F(I))$ and $\text{OPT}(I) - m(G(I, s') \leq \text{OPT}(F(I)) - m(s')$. As for the other direction, we can give a reduction similar to the one above from MAX SOL(Γ) to MAX SOL$(f(\Gamma))$ mapping optimal solutions back to optimal solutions. Hence, if MAX SOL$(f(\Gamma))$ is in **PO**, then so is MAX SOL(Γ). □

The concept of a core of a constraint language Γ has previously shown its value when classifying the complexity of CSP(Γ). We define a related concept for MAX SOL(Γ) and call it *max-core*.

Definition 7. A constraint language Γ is a max-core if and only if there is no non-injective unary operation f in $Pol(\Gamma)$ such that $f(d) \geq d$ for all $d \in D$. A constraint language Γ' is a max-core of Γ if and only if Γ' is a max-core and $\Gamma' = f(\Gamma)$ for some unary operation $f \in Pol(\Gamma)$ such that $f(d) \geq d$ for all $d \in D$.

The next lemma follows directly from Lemma 6.

Lemma 8. *If* Γ' *is a max-core of* Γ *and if* MAX SOL(Γ') *is* **APX**-*complete, then* MAX SOL(Γ) *is* **APX**-*hard and if* MAX SOL(Γ') *is in* **PO** *then so is* MAX SOL(Γ).

4 Approximability of MAX SOL

In this section present sufficient conditions for when MAX SOL is tractable and prove that it is **APX**-hard otherwise. To do so, we need a family of operations $\max_u : D^2 \to D$, $u \in D$, defined such that

$$\max_u(a, b) = \begin{cases} u & \text{if } \max(a, b) \leq u \\ \max(a, b) & \text{otherwise} \end{cases}$$

Theorem 9. Max Sol(Γ_L) *is tractable if* Γ_L *is invariant under* $\max_u$ *for some* $u \in D$. *Otherwise,* Max Sol(Γ_L) *is* **APX**-*hard.*

We divide the proof into three parts which can be found in Sections 4.1-4.3.

4.1 Tractability Result

Before we can prove the tractability of Max Sol$(Inv(\max_u))$, we need to introduce some terminology: Let $\mathbf{x} = (x_1, \ldots, x_r)$ be a list of r variables and let $(R, \mathbf{x})$ be a constraint on $\mathbf{x}$. For any sublist $\mathbf{x}' = (x_{i_1}, \ldots x_{i_k})$ of $\mathbf{x}$, define the *projection* of $(R, \mathbf{x})$ onto $\mathbf{x}'$, denoted $\pi_{\mathbf{x}'}(R, \mathbf{x})$, as follows: $\pi_{\mathbf{x}'}(R, \mathbf{x}) = \{\text{assignments to } (x_{i_1}, \ldots, x_{i_k}) \mid (R, (x_1, \ldots, x_r)) \text{ has a solution}\}$. A Csp instance is said to be *pair-wise consistent* if for any pair of constraints $(R, \mathbf{x}), (R', \mathbf{y})$, $\pi_{\mathbf{x}\cap\mathbf{y}}((R, \mathbf{x})) = \pi_{\mathbf{x}\cap\mathbf{y}}((R', \mathbf{y}))$.

Lemma 10. *If* Γ_L *is invariant under* $\max_u$ *for some* $u \in D$, *then* Max Sol(Γ_L) *is in* **PO**.

Proof. We begin by observing that if Γ_L is invariant under $\max_u$, then Γ_L is also invariant under the unary operation $u(x) = \max_u(x, x)$ (satisfying the condition $u(x) \geq x$). Hence, by Lemma 6, Max Sol(Γ_L) is in **PO** if Max Sol(Γ'_L) is in **PO** where $u(\Gamma_L) = \Gamma'_L$. Now, Γ'_L contains no tuple with an element $a < u$ and hence Γ'_L is invariant under $\max$ (since it is invariant under $\max_u$ and $\max_u$ acts as $\max$ on elements $\geq u$). Hence, it is sufficient to give a polynomial-time algorithm solving Max Sol(Γ'_L) for max-closed constraint languages Γ'_L. This algorithm is a straightforward modification of the polynomial-time algorithm for CSP$(Inv(\max_D))$ presented in [8].

Let $I = (V, D, C, w)$ be an instance of Max Sol(Γ'_L). Assume I to be pair-wise consistent. If I contains the empty constraint, then I has no solution. Otherwise, define $f : V \to D$ such that $f(x_i) = \max\{\pi_{(x_i)}((R, \mathbf{x})) \mid (R, \mathbf{x}) \in C\}$, i.e., f assigns to each variable the maximum value it is allowed by any constraint. We claim that this f is a solution to I. Consider any constraint $(R, \mathbf{x})$ where, say for simplicity, $\mathbf{x} = (x_1, \ldots, x_r)$. For each variable x_j, we must have some tuple $t_i \in R$ such that $t_i[j] = f(x_j)$ by pair-wise consistency and the choice of f. Since R is closed under $\max$, the maximum of all these tuples belong to R. This maximum tuple equals $(f(x_1), \ldots, f(x_r))$ and f satisfies the constraint.

Assume now that there exists a function $f' : V \to D$ such that f' satisfies all constraints in C and $\sum_{i=1}^{n} w(x_i) \cdot f'(x_i) > \sum_{i=1}^{n} w(x_i) \cdot f(x_i)$. Since $w(x_i) \geq 0$ for every $x_i \in V$, this implies that there exists at least one variable x_i such that $f'(x_i) > f(x_i)$. However, this is impossible by the choice of f.

To conclude the proof, f can be constructed in $O(|C|^2 a^2)$ time (where a is the maximum arity of the constraints in C) by Corollary 4.3 in [8]. □

4.2 APX-Hardness Results

We will show that whenever P is a negative pattern containing at least two literals, then Max Sol$(Rel(P))$ is **APX**-hard. We begin by presenting an **APX**-hardness result for the pattern $(-0, -1)$ over the domain $D = \{0, 1, 2\}$. The reduction is based on the well known **APX**-complete maximisation problem Max-E3Sat-5:

Instance: Set U of variables, collection C of disjunctive clauses containing exactly 3 literals each, and where each variable occurs at most 5 times.
Solution: A truth assignment for U.
Measure: Number of clauses satisfied by the truth assignment.

Lemma 11. *Let* $D = \{0, 1, 2\}$ *and* $r = \{(x, y) \in D^2 \mid x \leq 0 \vee y \leq 1\}$*, Then,* MAX SOL$(r)$-11 *is* **APX**-*complete.*

Proof. Membership in **APX** follows from the fact that the all-1 assignment is a 2-approximation. We prove **APX**-hardness by giving a L-reduction (with $\beta = 14$ and $\gamma = 1$) from MAX-E3SAT-5 to MAX SOL(r). The reduction relies on the following 'gadget': Let $V = \{A, B, C, a, b, c\}$ be a set of variables and impose the following constraints:

$$r(A, B), r(B, C), r(C, A), r(A, a), r(B, b), r(C, c).$$

One can see that $\max\{\sum_{v \in V} M(v) \mid M \text{ is a satisfying assignment}\} = 7$ and the optimum appears if and only if exactly one of A, B, C is assigned the value 2.

Let I be an arbitrary MAX-E3SAT-5 instance with m clauses $C_1, \ldots, C_m$. Construct a MAX SOL(r) instance $F(I) = (X, D, C, w)$ as follows:

$$X = \{X_1^1, X_2^1, X_3^1, x_1^1, x_2^1, x_3^1, \ldots, X_1^m, X_2^m, X_3^m, x_1^m, x_2^m, x_3^m\},$$

$w(x) = 1$ for all $x \in X$, and introduce a gadget on $X_1^i, X_2^i, X_3^i, x_1^i, x_2^i, x_3^i$ (as defined above) for each clause $C_i = \{l_1^i, l_2^i, l_3^i\}$. Finally, the clauses are connected by adding the constraints $r(X_j^i, X_{j'}^{i'})$ and $r(X_{j'}^{i'}, X_j^i)$ whenever $l_j^i = \neg l_{j'}^{i'}$.

By well-known arguments, at least half of the clauses in an instance of MAX-E3SAT-5 can be satisfied so $m \leq 2\text{OPT}(I)$. We also know that $\text{OPT}(F(I)) \leq 7m$ since each gadget corresponding to a clause contributes at most 7 to the measure of any solution to $F(I)$. It follows that $\text{OPT}(F(I)) \leq 14 \cdot \text{OPT}(I)$ and we can choose $\beta = 14$.

Now, given $F(I)$ and a solution s to $F(I)$, let $s' = G(F(I), s)$ be the solution to I (the instance of MAX-3SAT) defined as follows: $s'(x) = true$ if there exists a literal $l_j^i = x$ and $s(X_j^i) = 2$, $s'(x) = false$ if there exists a literal $l_j^i = \neg x$ and $s(X_j^i) = 2$, and $s'(x) = false$ for all other variables x. First we note that $s'(x)$ is well-defined; any two contradictory literals are prevented from being assigned the same truth value by the constraints introduced in the last step in the construction of $F(I)$.

We will show that $\text{OPT}(I) - m(I, s') \leq \text{OPT}(F(I)) - m(F(I), s)$ and $\gamma = 1$ is a valid parameter in the L-reduction. We begin by showing that $\text{OPT}(F(I)) - \text{OPT}(I) \geq 6m$. If $\text{OPT}(I) = k$, i.e., k clauses (but no more) can be satisfied, then each of the k satisfied clauses contains a true literal l_j^i. In each of the satisfied clauses C_i we choose one true literal (say l_j^i) and assign 2 to the corresponding variable X_j^i in the corresponding gadget G_i (on variables $\{X_1^i, X_2^i, X_3^i, x_1^i, x_2^i, x_3^i\}$) in $F(I)$. Assign 1 to x_j^i, 2 to the other two x^i variables, and 0 to the two unassigned X^i variables. In each gadget G_j corresponding to an unsatisfied clause C_j (in $\text{OPT}(I)$), assign 0 to all the X^j variables and 2 to all the x^j variables. The resulting solution to $F(I)$ shows that

$$\text{OPT}(F(I)) \geq 7k + 6(m - k) = k + 6m$$

and $\text{OPT}(F(I)) - \text{OPT}(I) \geq 6m$ since $k = \text{OPT}(I)$. Assume now that

$$\text{OPT}(I) - m(I, s) > \text{OPT}(F(I)) - m(F(I), s'),$$

or, equivalently, $m(F(I), s') - m(I, s) > 6m$. It is easy to reach a contradiction from this so $\text{OPT}(I) - m(I, s) \leq \text{OPT}(F(I)) - m(F(I), s')$ and $\gamma = 1$ is a valid parameter in the L-reduction. Also note that no variable occurs more than 11 times in the resulting instance $F(I)$ of MAX SOL(r). □

By combining the notion of max-cores and Lemma 11, we can prove **APX**-hardness for all negative patterns of length at least two:

Lemma 12. *If* $(-c_1, \ldots, -c_k) \in L$, $k \geq 2$, *then* MAX SOL(Γ_L) *is* **APX**-*hard.*

4.3 Proof of Theorem 9

Proof. Arbitrarily choose a clausal language L. If a pattern $(-c_1, -c_2, \ldots, -c_k)$, $k \geq 2$, exists in L, then MAX SOL(Γ_L) is **APX**-hard by Lemma 12. Hence, we can assume that for each $P \in L$ such that $|P| \geq 2$, it holds that P contains at least one positive literal. If all patterns in L are of length 1, then MAX SOL(Γ_L) is tractable since Γ_L is invariant under the operation max. Thus, we assume that L contains at least one pattern of length strictly greater than one. Let

$$u = \min\{\max U \mid U \subseteq D \text{ is definable by a pp-formula over } \Gamma_L\}$$

Let ψ be a pp-formula defining the set U, i.e., $U(x) \equiv \exists \mathbf{x} : \psi(x; \mathbf{x})$ and $\max U = u$. If there exists a pattern $(+a_1, \ldots, +a_p, -b_1, \ldots, -b_q)$, $q \geq 2$, and $a_i > u$ for all i, then there is a pp-formula that implements the relation $Rel((-b_1, \ldots, -b_q))$:

$$(y_1 \leq b_1 \vee \ldots \vee y_q \leq b_q) \equiv_{pp}$$

$$\exists z : (z \geq a_1 \vee \ldots \vee z \geq a_p \vee y_1 \leq b_1 \vee \ldots \vee y_q \leq b_q) \wedge U(z)$$

so MAX SOL(Γ_L) is **APX**-hard by Lemmata 4 and 12. If this is not the case, then we show that Γ_L is invariant under $\max_u$ and, by Lemma 10, that MAX SOL(Γ_L) is tractable. Arbitrarily choose a pattern $P \in L$. If $|P| \geq 2$, then we have two cases: Assume first that $P = (+a_1, \ldots)$ for some $a_1 \leq u$. Since $\max_u(a, b) \geq u$ for all choices of a, b, $Rel(P)$ is invariant under $\max_u$. Otherwise, $P = (+a_1, \ldots, +a_p, -b_1)$ and $a_i > u$ for all i. We see that $b_1 \geq u$ by the definition of u since $Rel((-b_1))$ can be implemented by a pp-formula:

$$(y_1 \leq b_1) \equiv_{pp} \exists z : (z \geq a_1 \vee \ldots \vee z \geq a_p \vee y_1 \leq b_1) \wedge U(z)$$

Arbitrarily choose two tuples $(t_1, \ldots, t_{p+1})$, $(t'_1, \ldots, t'_{p+1})$ from $Rel(P)$. If there exists a t_i, $1 \leq i \leq p$, such that $t_i \geq a_i$, then $(\max_u(t_1, t'_1), \ldots, \max_u(t_{p+1}, t'_{p+1}))$ is in $Rel(P)$. The situation is analogous if there exists a t'_i, $1 \leq i \leq p$, such that $t'_i \geq a_i$. Assume now that for all $1 \leq i \leq p$, $t_i < a_i$ and $t'_i < a_i$. This implies that $t_{p+1} \leq b_1$ and $t'_{p+1} \leq b_1$. If $\max(t_{p+1}, t'_{p+1}) \leq u$, then $\max_u(t_{p+1}, t'_{p+1}) = u$ and $Rel(P)$ is invariant under $\max_u$ since $b_1 \geq u$. If $\max(t_{p+1}, t'_{p+1}) > u$, then $\max_u(t_{p+1}, t'_{p+1}) = \max(t_{p+1}, t'_{p+1})$ and $Rel(P)$ is invariant under $\max_u$ also in this case.

We are left with the unary patterns in L. Assume that $P = (+r)$ for some r; in this case, $Rel(P)$ is trivially invariant under $\max_u$. If $P = (-r)$, then r must satisfy $r \geq u$ by the definition of u. Arbitrarily choose two elements $a, b \in (-r)$. If $\max(a,b) \leq u$, then $\max_u(a,b) = u$ and $Rel(P)$ is invariant under $\max_u$ since $r \geq u$. If $\max(a,b) > u$, then $\max_u(a,b) = \max(a,b)$ and $Rel(P)$ is invariant under $\max_u$. □

By inspecting the previous proof, we see that a constraint language Γ_L is closed under $\max_u$, $u \in D$, if and only if each pattern $P \in L$ satisfy at least one of the following conditions: (1) $P = (+a_1, ...)$ and $a_1 \leq u$; (2) $P = (+a_1, \ldots, +a_p, -b_1)$, $a_1, \ldots, a_p > u$, and $b_1 \geq u$; (3) $P = (+a)$; or (4) $P = (-a)$ and $a \geq u$. This makes it easy to check whether MAX SOL(Γ_L) is tractable or not: test if the condition above holds for some $u \in D$. If so, MAX SOL(Γ_L) is tractable and, otherwise, MAX SOL(Γ_L) is **APX**-hard by Theorem 9. Obviously, this test can be performed in polynomial time in the size of L and D. A simple algorithm that is polynomial in the size of Γ_L[2] also exists, but note that Γ_L can be exponentially larger than L and D.

5 Approximability of MIN SOL and MAX AW SOL

We now turn our attention to the two problems MIN SOL (i.e., the minimization version of MAX SOL) and MAX AW SOL (i.e., MAX SOL without the restriction of non-negative weights). We see, for instance, that MIN SOL($(+1, +1)$) (over the domain $D = \{0, 1\}$) is the same problem as the minimum vertex cover problem.

Obviously, the tractability results for MAX SOL can be transferred to the MIN SOL setting with only minor modifications: If Γ_L is invariant under $\min_u$ for some $u \in D$, then MIN SOL(Γ_L) is in **PO**. The operations $\min_u$ and $\max_u$ are symmetrically defined. By combining this with certain hardness results, one can prove the following:

Theorem 13. MIN SOL(Γ_L) *is in* **PO** *if* Γ_L *is invariant under* $\min_u$ *for some* $u \in D$. *Otherwise,* MIN SOL(Γ_L) *is* **APX**-*hard.*

Note that it easy to check whether MIN SOL(Γ_L) is tractable or not: the algorithm is similar to the algorithm for checking tractability of MAX SOL(Γ_L).

We continue by presenting sufficient and necessary conditions for tractability of MAX AW SOL.

Theorem 14. MAX AW SOL(Γ_L) *is in* **PO** *if* Γ_L *is invariant under both* max *and* min. *Otherwise,* MAX SOL(Γ_L) *is* **APX**-*hard.*

The tractability part is based on supermodular optimisation [13] while the inapproximability results are proved by appropriate reductions from MAX SOL and MIN SOL.

6 Conclusions and Open Questions

We have presented dichotomy results for the approximability of MAX SOL, MIN SOL, and MAX AW SOL when they are restricted to constraint languages expressed by regular signed logic. The results were partly obtained by exploiting certain algebraic methods that have previously not been widely used for studying optimisation problems.

[2] The size of a constraint language Γ over domain D is roughly $\sum_{R \in \Gamma} |R| \cdot \log |D| \cdot ar(R)$.

One way to extend this work is to provide a more fine-grained approximability analysis of these problems. In the case of boolean domains, such an analysis has been performed by Khanna et al. [10]; they prove that for any choice of allowed relations, the problem is either (1) polynomial-time solvable, (2) **APX**-complete, (3) **poly-APX**-complete, (4) finding a solution of measure > 0 is **NP**-hard; or (5) finding any solution is **NP**-hard. Another venue for future research would be investigate the approximability of MAX (AW) SOL(Γ) for arbitrary finite domain constraint languages Γ, i.e., constraint languages not necessarily expressed by regular signed logic.

References

1. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, 1999.
2. N. Creignou, M. Hermann, A. Krokhin, and G. Salzer. Complexity of clausal constraints over chains. 2006. To appear in: Theory of Computing Systems. Preliminary version available from: `www.cis.syr.edu/∼royer/lcc/LCC05`.
3. N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2001.
4. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
5. À.J. Gil, M. Hermann, G. Salzer, and B. Zanuttini. Efficient algorithms for constraint description problems over finite totally ordered domains. In *Proceedings of Automated Reasoning, Second International Joint Conference (IJCAR-04)*, pages 244–258, 2004.
6. R. Hähnle. Complexity of many-valued logics. In *Proceedings of the 31st IEEE International Symposium on Multiple-valued Logic (ISMVL-01)*, pages 137–148, 2001.
7. J.N. Hooker and M. Osorio. Mixed logical-linear programming. *Discrete Applied Mathematics*, 96-97:395–442, 1999.
8. P. G. Jeavons and M. C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79:327–339, 1996.
9. P. Jonsson. Boolean constraint satisfaction: complexity results for optimization problems with arbitrary weights. *Theoretical Computer Science*, 244(1-2):189–203, 2000.
10. S. Khanna, M. Sudan, L. Trevisan, and D.P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000.
11. R. E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.
12. R. Pöschel and L. Kalužnin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979.
13. A. Schrijver. A combinatorial algorithm minimizing submodular functions in polynomial time. *Journal of Combinatorial Theory, ser. B*, 80:346–355, 2000.
14. S.A. Wolfman and D.S. Weld. The LPSAT engine & its application to resource planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 310–317, 1999.

Probabilistic Length-Reducing Automata

Tomasz Jurdziński

Institute of Computer Science, Wrocław University,
Przesmyckiego 20, PL-51-151 Wrocław, Poland
tju@ii.uni.wroc.pl

Abstract. Hardness of a separation of nondeterminism, randomization and determinism for polynomial time computations motivates the analysis of restricted models of computation. Following this line of research, we consider randomized length-reducing two-pushdown automata (lrTPDA), a natural extension of pushdown automata (PDA). We separate randomized lrTPDAs from deterministic and nondeterministic ones, and we compare different modes of randomization. Moreover, we prove that amplification is impossible for Las Vegas automata.

1 Introduction

A comparative study of the computational power of deterministic, randomized and nondeterministic computations is one of the central topics in complexity and algorithm theory. Hardness of this problem for polynomial time computations motivated considerations of restricted models. This line of research started with the study of randomization for finite automata and continues by investigating more and more complex models (multicounter machines, communication protocols, OBDDs, PRAMs, distributed computing, etc.).

In last years interesting separation results were obtained for pushdown automata (PDA) [2,8,6]. As (deterministic) PDAs characterize (deterministic) context free languages ((D)CFL), we start the study of a machine model characterizing natural generalizations of CFL. Namely, we consider a machine model for growing context-sensitive languages (GCS), the class defined by grammars with growing rules, recognizable in polynomial time [3]. GCS is equal to the set of languages accepted by so-called length-reducing two-pushdown automata (lrTPDA) [1,11]. Such automaton consists of two pushdowns, an input word is given as the contents of one of them. One step of the computation takes k topmost symbols from both pushdowns and stores some shorter word instead. Intuitively, this model expands the power of PDAs into possibility of ,,moving" through the input many times. This ability is restricted by the fact that each step shortens the configuration. Deterministic variant of these automata characterize Church-Rosser Languages (CRL), the class that possesses natural characterization by string rewriting systems [9,11] (we denote CRL as DGCS here). The classes GCS and DGCS naturally complement the Chomsky hierarchy as they fill the gap between (D)CFL and (D)CSL.

The paper is organized as follows. In Section 2 we introduce some basic definitions. Section 3 describes formal tools used in our proofs. In Section 4 we compare Las Vegas automata with deterministic automata and we show that amplification is impossible for Las Vegas automata. Section 5 describes the results concerning the comparison of different modes of randomized lrTPDAs. Due to limited space, many details of the proofs are omitted.

2 Preliminaries

For a word x, let $|x|$, $x[i]$ and $x[i,j]$ denote the length of x, the ith symbol of x and the subword $x[i]\dots x[j]$ resp. Furthermore, let $[i,j]=\{l \mid i\le l\le j\}$, let x^R denote the reverse of the word x.

Let $x=x_1y_1x_2y_2\dots x_ny_nx_{n+1}$, $n>0$, where $x_i,y_i\in\Sigma^*$ for the alphabet Σ, $i\in[1,n+1]$, and y_i denotes the leftmost occurrence of y_i as a subword of x in the above factorization of x. Then, $x-(y_1,\dots,y_n)=x_1x_2\dots x_{n+1}$, and $x-(y_1,\dots,y_n)+(z_1,\dots,z_n)=x_1z_1x_2z_2\dots x_nz_nx_{n+1}$.

We use the notion of Kolmogorov complexity $K(x)$ of a word $x\in\{0,1\}^*$, and conditional Kolmogorov complexity $K(x|y)$ [7].

Fact 1. *[7] 1. Let X be a set of words such that $|X|\ge m$. Then, for each word w, there exists $x\in X$ such that $K(x|w)\ge\lfloor\log m\rfloor-1$.*
2. If $K(x_1x_2x_3|y)\ge n-p$, then $K(x_2|yx_1x_3)\ge|x_2|-p-O(\log n)$, where $n=|x_1x_2x_3|$.
3. $K(x)-K(x|y)=K(y)-K(y|x)\pm O(\log\max(|x|,|y|))$ for each $x,y\in\{0,1\}^*$.

A two-pushdown automaton $M=(Q,\Sigma,\Gamma,q_0,\perp,F,\delta)$ with a window of length $k=2j$ is a nondeterministic automaton with two pushdown stores, defined by the set of states Q, the input alphabet Σ, the tape alphabet Γ ($\Sigma\subseteq\Gamma$), the initial state $q_0\in Q$, the bottom marker of the pushdown stores $\perp\in\Gamma\backslash\Sigma$, the set of accepting states $F\subseteq Q$ and the transition relation $\delta:Q\times\Gamma_{\perp,j}\times\Gamma_{j,\perp}\to\mathcal{P}(Q\times\Gamma^*\times\Gamma^*)$, where $\Gamma_{\perp,j}=\Gamma^j\cup\{\perp v: |v|\le j-1, v\in\Gamma^*\}$, $\Gamma_{j,\perp}=\Gamma^j\cup\{v\perp: |v|\le j-1, v\in\Gamma^*\}$, $\mathcal{P}(Q\times\Gamma^*\times\Gamma^*)$ denotes the set of finite subsets of $Q\times\Gamma^*\times\Gamma^*$. The automaton M is *deterministic* if δ is a (partial) function from $Q\times\Gamma_{\perp,j}\times\Gamma_{j,\perp}$ into $Q\times\Gamma^*\times\Gamma^*$. M is called *length-reducing* (lr(D)TPDA) if $(p,u',v')\in\delta(q,u,v)$ implies $|u'v'|<|uv|$.

A *configuration* of M is described by a word uq_iv^R, where q_i is the current state, $u,v\in\Gamma^*$ are the contents of the pushdown stores. The transition $\delta(q,y,z)=(q',y',z')$ will be described as $yqz^R\to y'q'(z')^R$. For an input word $x\in\Sigma^*$, the corresponding *initial configuration* is $\perp q_0x\perp$, i.e., x is given as the contents of the second pushdown store. The automaton M *finishes* its computation by empty pushdown stores. So, $L(M)=\{x\in\Sigma^*:\exists_{q\in F}\ \perp q_0x\perp\vdash^*_M q\}$. We also require that the special symbol $\perp$ occurs only on bottoms of the pushdowns and no other symbol can occur on the bottom. The language classes defined by non-deterministic and deterministic lrTPDAs are denoted as GCS and DGCS, respectively.

A randomized or probabilistic lrTPDA is a (nondeterministic) lrTPDA with a probability distribution assigned to every nondeterministic branching (cf. [2]). The probability of the computation $\mathcal{C}$ is equal to the product of probabilities of the transitions executed in $\mathcal{C}$. The set of states is divided into accepting, rejecting and neutral states. Various modes of randomized automata considered in the literature differ in conditions concerning the required probability of finishing in the accepting, rejecting or the neutral state. We present these conditions in the following table, where ε denotes the probability of error.

	$w \in L$			$w \notin L$		
	accept	reject	neutral	accept	reject	neutral
Las Vegas (LV)	$\geq 1-\varepsilon$	0	$\leq \varepsilon$	0	$\geq 1-\varepsilon$	$\leq \varepsilon$
Monte Carlo (MC)	$\geq 1-\varepsilon$	$\leq \varepsilon$	0	0	1	0
bounded-error (B)	$\geq 1-\varepsilon$	$\leq \varepsilon$	0	$\leq \varepsilon$	$\geq 1-\varepsilon$	0

Abbreviations $\mathsf{LVGCS}_\varepsilon$, $\mathsf{MCGCS}_\varepsilon$ and $\mathsf{BGCS}_\varepsilon$ denote the sets of languages recognized by Las Vegas, Monte Carlo and bounded-error lrTPDAs with error probability ε. Furthermore, $X = \bigcup_{0<\varepsilon<1} X_\varepsilon$ for $X \in \{\mathsf{LVGCS}, \mathsf{MCGCS}\}$ and $\mathsf{BGCS} = \bigcup_{0<\varepsilon<1/2} \mathsf{BGCS}_\varepsilon$. It follows directly from the definitions that $\mathsf{DGCS} \subseteq \mathsf{LVGCS} \subseteq \mathsf{MCGCS} \subseteq \mathsf{GCS}$ and $\mathsf{MCGCS} \subseteq \mathsf{BGCS}$.

Our Results. We separate classes defined by various modes of probabilistic lrTPDAs from each other as well as from DGCS and GCS.

Theorem 1. $\mathsf{DGCS} \subsetneq \mathsf{LVGCS} \subsetneq \mathsf{MCGCS} \subsetneq \mathsf{BGCS}$, *moreover* $\mathsf{MCGCS} \subsetneq \mathsf{GCS}$.

Theorem 2. *For each $\varepsilon > 0$, there exists a language L such that $L \in \mathsf{BGCS}_\varepsilon \setminus \mathsf{GCS}$ and $\overline{L} \in \mathsf{MCGCS}_\varepsilon \setminus \mathsf{LVGCS}$.*

Finally it turns out that, despite many other models, amplification for Las Vegas lrTPDAs is not possible:

Theorem 3. $\mathsf{LVCF}_{1/2} \setminus \bigcup_{\alpha<1/2} \mathsf{MCGCS}_\alpha \neq \emptyset$.

3 Lower Bounds Tools

Let $M = (Q, \Sigma, \Gamma, q_0, \perp, F, \delta)$ be an lrTPDA with a window of length k. Each computation of M corresponds to a planar directed acyclic *computation graph* with the labeling ω, where ω is the function from the set of vertices to $\Gamma \cup Q \cup \delta$ (see examples in the full version of [4]). Vertices labeled with symbols, states, and transitions are called symbol vertices, state vert., and transition vert., resp. The *computation graph* $G_{(j)} = (V_j, E_j)$ corresponding to the computation $C_0 \vdash \ldots \vdash C_j$ (where C_0 denotes an initial configuration) is defined inductively:
Case 1: $j = 0$. Let $C_0 = \perp q_0 x_1 x_2 \ldots x_n \perp$. Then $G_{(0)} = (V_0, E_0)$, where $E_0 = \emptyset$, $V_0 = \{\rho_i\}_{i=-2}^{n+2}$ such that $\omega(\rho_i) = x_i$ for $1 \leq i \leq n$, $\omega(\rho_i) = \perp$ for $i \in \{-2, -1, n+1, n+2\}$ and $\omega(\rho_0) = q_0$.
Case 2: $j > 0$. Assume that the computation $C_0 \vdash_M \ldots \vdash_M C_{j-1}$ corresponds to the graph $G_{(j-1)}$, and the transition $z \to z'$ is executed in $C_{j-1} \vdash C_j$, where $|z| = p$ and $|z'| = p'$. The graph $G_{(j)}$ is constructed from $G_{(j-1)}$ by adding:

- vertices $\pi'_1, \ldots, \pi'_{p'}$ which correspond to the word z', $\omega(\pi'_i) = z'[i]$ for $i \in [1, p']$;
- a vertex D_j which corresponds to the transition $z \to z'$;
- edges $(\pi_1, D_j), \ldots, (\pi_p, D_j)$, where the vertices $\{\pi_i\}_{i=1}^{p}$ correspond to z;
- edges $(D_j, \pi'_1), \ldots, (D_j, \pi'_{p'})$.

There is a natural left to right ordering among the sources of the computation graph, induced by the left to right ordering into the initial configuration. There is also a left to right ordering among the in-neighbors and out-neighbors of each transition vertex. These orderings induce a left to right ordering among the sinks.

Note that *sources* (sinks, resp.) of $G_{(j)}$ correspond to the initial configuration C_0 (the last configuration C_j), the sequence of their labels will be denoted as $src(G_{(j)})$ ($snk(G_{(j)})$, resp.). The single step transition relation $\vdash_M$ on configurations can be extended to computation graphs, i.e., $G_{(j-1)} \vdash_M G_{(j)}$ for $G_{(j)}$, $G_{(j-1)}$ defined above.

The term *path* is applied exclusively to paths that start in a source vertex and finish in a sink. The ordering among vertices of the graph induces a left-to-right partial ordering of paths. A path σ_1 is to the left of a path σ_2 iff none of the vertices of σ_1 is to the right of any vertex of σ_2. A path σ is the leftmost (rightmost) in a set of paths S if it is to the left (right) of each other path $\sigma' \in S$.

Let σ be a path in G with a sink π. We say that σ is *short* if there is no path with the sink π that is shorter than σ. Let us note here that a sink of a graph may be the endpoint of short paths which start in different sources. On the other hand, it is possible that no short path starts in a particular source vertex.

Lemma 1. *[4] Let G be a computation graph.*

1. *Let V_{sr} and V_{sn} be subsets of the set of sources of G and the set of sinks of G, resp. Then, the set P of short paths with sources in V_{sr} and sinks in V_{sn} contains a path which is to the right/left of all other paths in P (or $P = \emptyset$).*
2. *The length of each short path in G is at most $c \log n$, where n is the length of the input word and the constant c depends only on the automaton M.*

Now, we introduce definitions needed to formulate cut and paste technique. A *description of a path* $\sigma = \pi_1, \pi_2 \ldots, \pi_{2l+1}$, denoted $\mathsf{desc}(\sigma)$, consists of the sequence $(\omega(\pi_1), p_1), \ldots (\omega(\pi_{2l+1}), p_{2l+1})$ such that π_i is the p_i-th in-neighbor of π_{i+1} (according to the left-to-right ordering) for odd $i < 2l$, and π_{i+1} is the p_{i+1}-st out-neighbour of π_i for even i, $p_{2l+1} = 0$. A *full description* of a path σ in a computation graph G, $\mathsf{descf}(\sigma)$, is equal to $(\mathsf{desc}(\sigma), p)$, where p is the position of the source vertex of σ in the initial configuration.

We say that descriptions $\gamma_1, \ldots, \gamma_{l-1}$ (for $l > 1$) *decompose* a computation graph G into subgraphs $G_1, \ldots, G_l$ if G contains paths $\{\sigma_i\}_{i=1}^{l-1}$ such that: $\mathsf{desc}(\sigma_i) = \gamma_i$ for $i \in [1, l]$, σ_i is located to the left of σ_{i+1} for $i \in [1, l-2]$, G_i is equal to the subgraph of G which contains all vertices and edges located between σ_{i-1} and σ_i (where σ_0 and σ_l are the "artificial" empty paths located to the left/right of all other vertices of G). If descriptions $\gamma_1, \ldots, \gamma_{l-1}$ decompose G into $G_1, \ldots, G_l$, we denote it as $G = G_1 \ldots G_l$.

Lemma 2 (Cut and Paste Lemma). *[4] Assume that the descriptions γ_1, γ_2 decompose graphs G, H into G_1, G_2, G_3 and H_1, H_2, H_3, resp. Then, $J = G_1 H_2 G_3$ is a computation graph.*

Let $G = G_1 \dots G_l$ $(l > 1)$. Then, $\mathsf{border}(G_i, G_{i+1})$ is equal to the path between G_i and G_{i+1}, and the *surface* of G_i, $\mathsf{srf}(G_i)$, is defined as $(\mathsf{desc}(\sigma_1), \mathsf{desc}(\sigma_2), \omega(snk(G_i)))$, where $\sigma_1 = \mathsf{border}(G_{i-1}, G_i)$ and $\sigma_2 = \mathsf{border}(G_i, G_{i+1})$.

Now, we describe the notion of *image*, which allows to determine how much information about a subword of the input word is stored in a configuration. Let G be a computation graph corresponding to the computation on $x = x_1x_2x_3$, let σ_1, and σ_2 be the rightmost short path with the source in $x_1x_2[1]$ and the leftmost short path with the source in $x_2[|x_2|]x_3$, resp. (such σ_1, σ_2 always exist by Lemma 1.1 and thanks to the vertices ρ_{-2} and ρ_{n+2}). Then, if σ_1 is not to the left of σ_2 or there is no symbol sink vertex between the sinks of σ_1 and σ_2, the *image* of x_2 in G is *undefined.* Otherwise, let $G = G_1G_2G_3$, where σ_1, σ_2 are the paths on the borders of G_2. Then, the image of x_2, $\mathsf{img}(x_2)$, is equal to $\mathsf{srf}(G_2)$, the length of the image is equal to the number of sinks of G_2(i.e., $|snk(G_2)|$) and we say that G_2 *defines* the image of x_2. We will often identify the image with the sinks and the paths on the borders of the subgraph which defines it. In Section 4 we implicitly use the following properties of images.

Proposition 1. *[4] Let G, G' be computation graphs corresponding to the computation on the input word $xyzvu$, $G \vdash G'$. Assume that $\mathsf{img}(y)$ is defined in G and it is equal to $(\sigma_1, \sigma_2, \tau)$. Then,*
(a) For every $1 < i < |\tau|$, $\tau[i]$ is the sink of a short path which starts in y.
(b) If $|\tau| > 2k$ then $\mathsf{img}(y)$ is defined in G', and its length is in $[|\tau| - k, |\tau| + k]$.
(c) If $\mathsf{img}(v)$ is defined in G and it is equal to $(\sigma'_1, \sigma'_2, \tau')$, then $|\tau \cap \tau'| \leq 2$.
(d) If $\mathsf{img}(y)$ is undefined in G, it is undefined in G' as well.

Moreover, we will make use of the following fact. If the image of x in a graph G is included in a subgraph G_1 and the paths on the borders of G_1 are short, then the image of x remains unchanged if we cut G_1 and paste it into another graph, provided that the paths on the borders of G_1 remain short.

Below, we present some technical notions and results, partly introduced in [5]. Let $v = x_1\, x_2\, x_3\, x_4\, x_5$ be an input word, let $\mathcal{C}$ be a computation on v. We say that the image of x_i is *short* if its length is $\leq \log |x_i|$ and it is *long* otherwise. Furthermore, we say that the pair (x_2, x_4) is *checked* during the computation $\mathcal{C}$, if $\mathsf{img}(x_2)$ and $\mathsf{img}(x_4)$ are long, as long as the image of y_2 is defined and longer than $2k$ (where k is equal to the size of the window).

Proposition 2. *Let $\mathcal{C}$ be a computation of an* lrTPDA *M on the input word $x = x_1x_2x_3x_4x_5x_6$, which finishes with empty pushdown stores. If the pair (x_2, x_4) is checked in $\mathcal{C}$, the pair (x_3, x_5) is not checked in $\mathcal{C}$.*

Assume that the pair (x_2, x_4) is not checked in the computation $\mathcal{C}$ on $x_1x_2x_3x_4x_5$. Then, the *critical graph* with respect to (x_2, x_4) in $\mathcal{C}$ is equal to the first graph in $\mathcal{C}$ (according to the relation $\vdash$) in which $\mathsf{img}(x_2)$ or $\mathsf{img}(x_4)$ is short. Using properties of images, one can show that the graphs defining the images of x_2 and x_4 are disjoint in the critical graph with respect to (x_2, x_4).

We associate "probabilities" to paths and subgraphs of computation graphs which describe computations of probabilistic lrTPDAs. The probability $P(\sigma)$ ($P(G)$, resp.) of a path σ (a subgraph G, resp.) is equal to the product of the probabilities of random choices associated to the transition vertices in σ (the transition vertices in G except the vertices on the borders of G, resp.).

4 Determinism, Las Vegas and Amplification Issues

Let $L_{(1)} = L_{1,1} \cup L_{1,2}$, where

$$L_{1,1} = \{x_1\#x_2\#x_3\#y_2\#y_3\#y_1 \mid (x_1 = y_1^R \wedge x_2 = y_2^R), x_i, y_i \in \{0,1\}^* \text{ for } i \in [1,3]\}$$
$$L_{1,2} = \{x_1\#x_2\#x_3\#y_2\#y_3\#y_1 \mid (x_1 \neq y_1^R \wedge x_3 = y_3^R), x_i, y_i \in \{0,1\}^* \text{ for } i \in [1,3]\}$$

We show that $L_{(1)} \in \mathsf{LVGCS}_{1/2}$ and $L_{(1)}$ cannot be accepted by any Las Vegas nor Monte Carlo lrTPDA with the probability of error $1/2 - \alpha$ for each constant $0 < \alpha \leq 1/2$. Thus, $L_{(1)}$ certifies that $\mathsf{DGCS} \subsetneq \mathsf{LVGCS}$ and amplification is impossible for Las Vegas and Monte Carlo lrTPDAs.

First, observe that $L_{(1)}$ can be recognized by a Las Vegas lrTPDA with error pbb 1/2. The automaton M chooses (a) or (b) with pbb 1/2:
(a) compare x_2 with y_2: if $x_2 = y_2^R$ and $x_1 = y_1^R$: accept; if $x_2 \neq y_2^R$ and $x_1 = y_1^R$: reject; if $x_1 \neq y_1^R$: neutral state; (b) compare x_3 with y_3: we leave details to the reader.

The remaining part of this section is devoted to the proof that $L_{(1)} \notin \mathsf{LVGCS}_{1/2-\alpha} \cup \mathsf{MCGCS}_{1/2-\alpha}$ for each $0 < \alpha \leq 1/2$.

4.1 Sketch of the Proof

In order to shorten notations, we will denote the input word $z_1\#z_2\#\ldots\#z_6$ as $z_1z_2\ldots z_6$ (i.e., the factorization $z_1\ldots z_6$ denotes the positions in which the symbol # appears). First, we need the following proposition, which can be proved by fooling the automaton M using cut and paste technique, combined with incompressibility method (cf. [7]).

Proposition 3. *Let M be a nondeterministic (or LV, MC) lrTPDA that recognizes $L_{(1)}$. Let $w = x_1x_2x_3y_2y_3y_1 \in L_{(1)}$, where $K(x_i|w - (x_i, y_i)) > m/2$, $K(y_i|w - (x_i, y_i)) > m/2$, $|x_i| = |y_i| = m$ for $i \in [1,3]$, and m is large enough. Then, for each accepting computation of M on w, we have:*
1. If $w \in L_{1,1}$ then M checks the pair (x_2, y_2).
2. If $w \in L_{1,2}$ then M checks the pair (x_3, y_3).

For the sake of contradiction, assume a LV (or MC) lrTPDA M recognizes $L_{(1)}$ with error probability $1/2 - \alpha$ for $0 < \alpha \leq 1/2$. Let $w = x_1x_2x_3y_2y_3y_1 \in L_{(1)}$, where $x_i = y_i^R$, $|x_i| = m$ for $i \in [1,3]$, $K(x_1x_2x_3) > 3m - 1$ and m is large enough. We define the set X of words such that M behaves "similarly" on w and on $w' = w - x_1 + x$ for each $x \in X$. That is, the pbb that M checks (x_3, y_3) on

w' and the pbb that M checks (x_3, y_3) on w differ by $o(1)$. As $w \in L_{1,1}$, M checks (x_2, y_2) in each accepting computation on w (Proposition 3). So, it does not check (x_3, y_3) in each accepting computation on w (Proposition 2) what implies that M does not check (x_3, y_3) on w with pbb $\geq 1/2 + \alpha$, and it does not check (x_3, y_3) on w' with pbb $1/2 + \alpha \pm o(1)$. On the other hand, the word $w' = w - x_1 + x \in L_{1,2}$ satisfies the assumptions of Prop. 3. So, M cannot accept w' when (x_3, y_3) is not checked. Thus, M does not accept $w' \in L_{(1)}$ with pbb $1/2 + \alpha \pm o(1) > 1/2$: contradiction. Below, we describe the construction of X more precisely.

We say that $x \in \{0,1\}^m$ is *similar* to x_1 iff the suffixes of length $2^{\sqrt{\log m}}$ of x and x_1 are equal. In the following, a *critical graph* stands for the critical graph with respect to (x_3, y_3), in a computation of M on w or on $w - x_1 + x$ for $x \in \{0,1\}^m$ which is similar to x_1. Let G be the critical graph which describes the computation on w. The key technical result in the proof states that the graph G has to contain a short path of length $\leq \sqrt{\log m}$ which starts in the suffix of x_1 of length $2^{\sqrt{\log m}}$ (Proposition 4). So, let Y be the set of all possible full descriptions of paths of length $\leq \sqrt{\log m}$, with the source vertex on the position in $[m - 2^{\sqrt{\log m}}, m]$. Note that the size of Y is $o(m/\log m)$. Using this fact we define a compact description of computations of M on w' which do not check the pair (x_3, y_3), where $w' = w - x_1 + x$ and x is similar to x_1. To this aim,

(a) We divide the set of critical graphs of M on w' into groups. Let G be a critical graph, let σ be the rightmost short path in G which starts in x and is not longer than $\sqrt{\log m}$. A group of the graph G is determined by $\gamma \in Y$, the full description σ. The path σ is called the *critical path*, the decomposition of G defined by σ is called the *critical decomposition.*
(b) For each full description $\gamma \in Y$, we define the set of left subgraphs $(\mathrm{Lf}(\gamma, w'))$ and right subgraphs $(\mathrm{Rg}(\gamma, w'))$ determined by the critical decompositions of critical graphs on w'.
(c) For each γ as above, let $P(\mathrm{Lf}(\gamma, w'))$ be the sum of the probabilities of all subgraphs in $\mathrm{Lf}(\gamma, w')$. The approximations of the numbers $P(\mathrm{Lf}(\gamma, w'))$ by $O(\log m)$ most significant bits for each $\gamma \in Y$ form the description $D_{w'}$.

Let $D = D_w$ be the description of the behavior of M on w. We say that $x \in \{0,1\}^m$ *agrees* with D iff the description $D_{w'}$ for $w' = w - x_1 + x$ is equal to D and x is similar to x_1. Observe that one can encode D using $o(m)$ bits.

Cut and Paste Lemma implies that one can combine any $G_1 \in \mathrm{Lf}(\gamma, w)$ with any $G_2 \in \mathrm{Rg}(\gamma, w)$ in order to obtain the computation graph $G = G_1G_2$. It turns out that the graph G obtained in this way is critical (Proposition 5). Moreover, each critical graph G on w has exactly one factorization G_1, G_2 such that $G_1 \in \mathrm{Lf}(\gamma, w)$ and $G_2 \in \mathrm{Rg}(\gamma, w)$ for $\gamma \in Y$. More importantly, the graph G_1G_2 is critical on w' for each $G_1 \in \mathrm{Lf}(\gamma, w)$, $G_2 \in \mathrm{Rg}(\gamma, w')$ and $w' = w - x_1 + x$, where x is similar to x_1. Moreover, each critical graph on w' has at most one factorization of this type (Proposition 6). Assume that x agrees with the description D. Thanks to the above properties, the probabilities that M does not check the pair (x_3, y_3) on w and on $w' = w - x_1 + x$ are very close. It might be the case that the probability of checking (x_3, y_3) is smaller on w' than on w, what follows from the fact that the numbers associated with the descriptions of the paths from Y do not represent the appropriate sums precisely (see (c)). However, we assure the precision which guarantees that the difference between these two probabilities is $o(1)$.

Let X be the set of words which agree with D. It turns out that one can determine X on the basis of D and $w-(x_1,y_1)$ only. Thus, knowing x_2,x_3, one can encode x_1 by D and the position of x_1 (in the lexicographic order) in X. Using properties of Kolmogorov complexity and the assumption that $K(x_1x_2x_2) > 3m-1$, we conclude that the size of X is large. (Otherwise, we could compress $x_1x_2x_3$ too much.) But this fact implies (by the application of Fact 1.3) that X contains a word $x \neq x_1$ such that $w-x_1+x$ satisfies assumptions of Proposition 3. So, the fraction of computations on $w-x_1+x$ which check (x_3,y_3) is too small.

4.2 Some Technical Details of the Proof

Proposition 4. *Let G be the critical graph with respect to (x_3,y_3), in an accepting computation $\mathcal{C}$ on w. Then, there exists a short path σ in G such that its length is $\leq \sqrt{\log m}$ and its source is in $x[m-2^{\sqrt{\log m}},m]$.*

Let the terms *critical graph, critical path, and critical decomposition* be defined as in the previous section. Moreover, as before, let Y be the set of all possible full descriptions of paths of length $\leq \sqrt{\log m}$, with the source vertex on the position in $[m-2^{\sqrt{\log m}},m]$. To be precise, we define $\mathrm{Lf}(\gamma,w')$ $(\mathrm{Rg}(\gamma,w'))$ for each $\gamma \in Y$ and $w' = w-x_1+x$ in the following way: G_1 (G_2, resp.) belongs to $\mathrm{Lf}(\gamma,w')$ ($\mathrm{Rg}(\gamma,w')$, resp.) iff there exists a critical graph $G = G_1G_2$ on w' such that the decomposition G_1,G_2 is critical and the full description of $\mathsf{border}(G_1,G_2)$ is equal to γ. Then,

Proposition 5. *1. Let $G \neq H$ be critical graphs on w, with respect to (x_3,y_3). Then, it is not the case that $G \vdash^* H$.*
2. Let $G_1 \in \mathrm{Lf}(\gamma,w)$, $G_2 \in \mathrm{Rg}(\gamma,w)$ for the full description $\gamma \in Y$. Then, G_1G_2 is the critical graph with respect to (x_3,y_3), and G_1,G_2 is its critical decomposition.

Let $P(\mathrm{Lf}(\gamma,w')) = \sum_{G_1\in\mathrm{Lf}(\gamma,w')} P(G_1)$ and $P(\mathrm{Rg}(\gamma,w')) = \sum_{G_2\in\mathrm{Rg}(\gamma,w')} P(G_2)$. Using the above proposition, we make the following calculations

$$\begin{aligned}\tfrac{1}{2}+\alpha &\leq P(M \text{ accepts } w)\\ &\leq \textstyle\sum_{G_1G_2 \text{ critical on } w \text{ w.r.t. } (x_3,y_3)} P(G_1G_2)\\ &= \textstyle\sum_{G_1G_2 \text{ critical on } w \text{ w.r.t. } (x_3,y_3)} P(G_1)P(\mathsf{border}(G_1,G_2))P(G_2)\\ &= \textstyle\sum_{\gamma\in Y} P(\gamma)P(\mathrm{Lf}(\gamma,w))\cdot P(\mathrm{Rg}(\gamma,w)),\end{aligned} \tag{1}$$

where the second inequality follows from Propositions 3.1 and 2, the last equality follows from Propositions 5.1 and 5.2.

Observe that the size of Y is $\leq 2^{c\sqrt{\log m}}$ for the constant c. Let D be a ***description*** of the behavior of M on w which consists of $x_1[m-2^{\sqrt{\log m}},m]$ and, for each $\gamma \in Y$, the $2c\sqrt{\log m}$-bit approximation of $P(\mathrm{Lf}(\gamma,w))$ together with the bit indicating whether $\mathrm{Lf}(\gamma,w)$ is empty. (The l-bit approximation of the number $a = \sum_{i=1}^{\infty} a_i 2^{-i}$ is defined as $\sum_{i=1}^{l} a_i 2^{-i}$, where $a_i \in \{0,1\}$ for each *i*.) The description D can be stored in $O(2^{c\sqrt{\log m}}) + O(2^{c\sqrt{\log m}} \cdot 2c\sqrt{\log m}) = o(m)$ bits. We say that a word $x \in \{0,1\}^m$ ***agrees*** with the description D if: **(i)** $x[m-2^{\sqrt{\log m}}, m] = x_1[m-2^{\sqrt{\log m}}, m]$, and **(ii)** the $2cp$-bits approximations of $P(\mathrm{Lf}(\gamma, w - x_1 + x))$, and $P(\mathrm{Lf}(\gamma, w))$ are equal for each $\gamma \in Y$ such that $\mathrm{Lf}(\gamma, w) \neq \emptyset$. Let X be the set of words from $\{0,1\}^m$ which agree with D.

Proposition 6. *Let $x \in X$, $w' = w - x_1 + x$. Then, for each $\gamma \in Y$, $G_2 \in \mathrm{Rg}(\gamma, w)$, and $G_1 \in \mathrm{Lf}(\gamma, w')$, G_1G_2 is the critical graph with respect to the pair (x_3, y_3); and G_1, G_2 is the critical decomposition of this graph.*

The key point for the application of the description D is that one can determine the set $\mathrm{Lf}(\gamma, w - x_1 + x)$ on the basis of x and D, independent of x_1, x_2, x_3.

Proposition 7. *For each $\gamma \in Y$ such that $\mathrm{Lf}(\gamma, w) \neq \emptyset$, and each $x \in \{0,1\}^m$ such that $x[m - 2^{\sqrt{\log m}}, m] = x_1[m - 2^{\sqrt{\log m}}, m]$ one can determine $\mathrm{Lf}(\gamma, w - x_1 + x)$ on the basis of x only.*

Proposition 7 implies that one can determine the set X on the basis of D only. As the size of D is $o(m)$, $x_1 \in X$ and $K(x_1x_2x_3) > 3m - 1$, it should be the case that $|X| \geq \Omega(2^{3m/4})$. Indeed, otherwise we could compress x_1 by its position in X. Using the fact that X is large, one can show that the assumptions of Proposition 3.2 are satisfied for some $w' = w - x_1 + x$, where $x \in X$ (we apply Fact 1.3 here). So, M should check (x_3, y_3) on w' with pbb $\geq 1/2 + \alpha$. However, we show that the behavior of M on w' is very close to its behavior on w (while M checks (x_3, y_3) on w with pbb $\leq 1/2 - \alpha$). So,

$$\begin{aligned}
P(\neg(M \text{ acc. } w')) &\geq P(M \text{ does not check } (x_3, y_3) \text{ on } w') \\
&\geq \textstyle\sum_{\gamma \in Y} \sum_{G_1 \in \mathrm{Lf}(\gamma, w')} \sum_{G_2 \in \mathrm{Rg}(\gamma, w)} P(\gamma) P(G_1) P(G_2) \\
&= \textstyle\sum_{\gamma \in Y} P(\gamma) \cdot P(\mathrm{Lf}(\gamma, w') \cdot P(\mathrm{Rg}(\gamma, w)) \\
&\geq \tfrac{1}{2} + \alpha - 2^{-2c\sqrt{\log m}} \cdot (\textstyle\sum_{\gamma \in Y} P(\gamma) \cdot P(\mathrm{Rg}(\gamma, w))
\end{aligned}$$

where the first three (in)equalities follow from Propositions 3, 2, 6. The last inequality follows from the fact that $P(\mathrm{Lf}(\gamma, w')) \geq P(\mathrm{Lf}(\gamma, w)) - 2^{-2c\sqrt{\log m}}$ for each $\gamma \in Y$ such that $\mathrm{Lf}(\gamma, w)) \neq \emptyset$ (because these two sums agree on first $2c\sqrt{\log m}$ bits), and

$$\textstyle\sum_{\gamma \in Y} P(\gamma) \cdot P(\mathrm{Lf}(\gamma, w)) \cdot P(\mathrm{Rg}(\gamma, w)) \geq \tfrac{1}{2} + \alpha$$

by Equation (1). Using Proposition 5, one can show that $P(\mathrm{Rg}(\gamma, w)) \leq 1$. Finally the probability that M does not accept $w' \in L_{1,2}$ is $\geq 1/2 + \alpha - 2^{-2c\sqrt{\log m}}$. $\sum_{\gamma \in Y} P(\gamma) \cdot 1 \geq 1/2$ for m large enough because $\sum_{\gamma \in Y} P(\gamma) \leq |Y| \leq 2^{c\sqrt{\log m}}$. This contradicts the assumption that M accepts $L_{(1)}$ with probability $\geq 1/2 + \alpha$.

5 Symmetric vs Asymmetric Modes of Acceptance

In this section we prove Theorem 2 which implies also some of the relationships of Theorem 1.

Lemma 3. *Let $L_{(3)}(n) = \{(w\#w^R\#)^n \mid w \in \{0,1\}^*\}$. Then, for each $\varepsilon > 0$, there exists n such that $\overline{L_{(3)}(n)} \in \mathsf{MCGCS}_\varepsilon \setminus \mathsf{LVGCS}$ and $L_{(3)}(n) \in \mathsf{BGCS}_\varepsilon \setminus \mathsf{GCS}$.*

Proof. The language $L_{(3)}(n)$ is not in GCS for each $n > 1$ what is the simple consequence of Lemma 2.2.6 in [10]. The relationship $\overline{L_{(3)}(n)} \notin$ LVGCS follows from the fact that LVGCS $\subseteq$ GCS and LVGCS is closed under complement.

Below, we describe the algorithm which can be implemented as lrTPDA. It tries to detect a pair of blocks w_i, w_j of the input word $w_1\# \ldots \# w_{2n}$ such that $w_i \neq w_j^R$ and $|i - j|$ is odd. Let $k = \log n$. The algorithm works as follows:

1. The set of natural numbers $p_1, p_2, \ldots, p_{k-1}$ from $[1, n]$ is chosen randomly (with uniform distribution). Further, the numbers $p_1, \ldots, p_{k-1}$ are sorted. So, let $p_1 \leq \ldots \leq p_{k-1}$.
2. Next, w is divided into k *segments*, the ith segment contains $r_i = 2(p_i - p_{i-1})$ blocks (i.e., $w_{2p_{i-1}+1} \ldots w_{2p_i}$), where $p_0 = 0$ and $p_k = 2n$.
3. For each $i \in [1, k]$, we check whether $w_{2p_{i-1}+r_i/2-j+1} = w^R_{2p_{i-1}+r_i/2+j}$ for $j = 1, \ldots, r_i/2$. If we find (in any segment) a pair of blocks which does not satisfy this condition, the algorithm rejects. Otherwise, it accepts.

The values $2p_1, \ldots, 2p_{k-1}$ are called *transition points.*

Observe that the above algorithm accepts each $w \in L_{(3)}(n)$. We show that the probability that this algorithm accepts $w \notin L_{(3)}(n)$ goes to zero when n goes to infinity. It will directly imply the second relationship of Lemma 3. The first relationship is obtained if one replaces the accepting/rejecting states.

We say that the word $w = w_1\#w_2^R\#w_3\# \ldots w_{2n-1}\#w_{2n}^R \notin L_{(3)}(n)$ is a *basic input* iff there exist $x_0 \neq x_1$ such that $w_i \in \{x_0, x_1\}$ for each $i \in [1, 2n]$.

Claim 1. *If the probability of error of the above algorithm is $\leq \varepsilon$ for all basic inputs, then the probability of error is bounded by ε for all inputs.*

Now, we analyze computations on the fixed basic input $w = w_1\#w_2^R\#w_3\ \#w_4^R \ldots \#w_{2n-1}\#w_{2n}^R$. Let x_0, x_1 be its "basic blocks", i.e., $w_i \in \{x_0, x_1\}$ for each $i \in [1, 2n]$. Let $\alpha \in \{0, 1\}^{2n}$ be a *characteristic vector* of w such that $\alpha_j = 0$ iff $w_i = x_0$. It induces the following interpretation of our algorithm. We split $\alpha \in \{0, 1\}^{2n}$ into $\log n$ consecutive subwords of even length. And, the question is what is the probability that each of the subwords is a palindrome for $\alpha \notin 0^{2n} + 1^{2n}$ if each partition of α is equally probable.

We split a characteristic vector α into *sectors.* The ith sector is equal to the longest prefix of the remaining part of α which belongs to $0^* + 1^*$. However, if this prefix is shorter than 6, we join it with the consecutive sequences from $0^* + 1^*$ into one sector until its length is equal or greater than 6 (see Example 1). Similarly, if the length of the rightmost sector is ≤ 6, it is joined with the last but one sector. (Observe that *sectors* are determined by the input word, while *segments* are determined by the random choice of the algorithm.)

For a fixed characteristic vector α which consists of l sectors, we split the probabilistic choices of the algorithm into groups. A *group* is defined by a binary vector $\mathcal{M} \in \{0, 1\}^l$ called a *map* such that $\mathcal{M}_i = 1$ iff at least one transition point belongs to the ith sector. (We assume that the transition point between the ith sector and the $(i+1)$st sector belongs to the ith sector.) We say that the sector i belongs to the map $\mathcal{M}$ iff $\mathcal{M}_i = 1$ (see Example 1).

Example 1. For the input word $x_0x_1^Rx_0x_2^Rx_3x_3^Rx_1x_2^Rx_0x_2^R$, we have

input word	x_0	x_1^R	x_0	x_2^R	x_3	x_3^R	x_1	x_2^R	x_0	x_2^R
basic input	x_0	x_1^R	x_0	x_1^R	x_1	x_1^R	x_1	x_1^R	x_0	x_1^R
characteristic vector	0	1	0	1	1	1	1	1	0	1

The characteristic vector α = 111111 0000000 11100000 110011111 is divided into the sectors 111111, 0000000, 11100000, and 110011111. As α consists of 4 sectors, the map of each partition of α into segments is the 4-bit vector. For example, the map $\mathcal{M} = 0110$ describes probabilistic choices which contain at least one transition point in the second sector, at least one transition point in the third sector and no transition points in the sectors 1 and 4. □

We show that the probability of error (i.e., the probability that $w \notin L_{(3)}$ is accepted) is $o(1)$ for *almost* each map. We ignore some "inconvenient" partitions (i.e., maps) chosen by the algorithm, but these partitions occur with the probability going to zero when $n \to \infty$.

We say that a sector of the characteristic vector is *short* if its length is $\leq \log^3 n$. Otherwise, the sector is *long*. We are going to avoid analysis of partitions in which more than one transition point occurs in any short sector. This is achieved by the following claim.

Claim 2. *The probability that there exists a short sector which contains more than one transition point goes to* 0 *when* $n \to \infty$.

Assuming that there exists at most one transition point in each short sector, we consider two cases:

Case 1. The map $\mathcal{M}$ contains at least $\frac{1}{2}\log n$ short sectors (i.e., $\mathcal{M}_i = 1$ for at least $\frac{1}{2}\log n$ i's such that the ith sector is short).

Note that the map $\mathcal{M}$ determines uniquely the number of transition points in short sectors. So, we can assume that the transition points in long sectors are chosen independently of the transition points in the short sectors. For each fixed choice of transition points in long sectors, we show that the probability of error is $o(1)$. Let $s_1, \ldots, s_{(\log n)/2}$ be the short sectors of $\mathcal{M}$, ordered from left to right. Let us consider any choice of the transition points in the sectors $S_{odd} = \{s_1, s_3, s_5, \ldots\}$. We show that at least 1/3 of choices of transition points in each sector of $S_{even} = \{s_2, s_4, s_6, \ldots\}$ allows to find non-fitting elements, independently of choices in other sectors from S_{even}. It is based on the following claim.

Claim 3. *Let* $x \in \{0,1\}^m$*, where* m *is even,* $x \neq 0^m$ *and* $x \neq 1^m$*. Then, for each* $r < m/2$ *at least one of the following two conditions is not satisfied:*

(a) $x[1, 2r]$ *and* $x[2r+1, m]$ *are palindromes,*
(b) $x[1, 2r+2]$ *and* $x[2r+3, m]$ *are palindromes.*

For the fixed transition points in the sectors of S_{odd}, the choice of the transition point in any sector of S_{even} is equivalent to the choice of a partition of a word $z \notin \{0^*, 1^*\}$ into two subwords (segments) z_1, z_2. Next, we check whether z_1 and z_2 are palindromes. Indeed, the two nearest transition points to s_{2i} are fixed (as they belong to long blocks or to blocks from S_{odd}). So, let z be the word between these two nearest transition points. Then, $z \notin 0^* + 1^*$. As each block is not shorter than 6, it contains at least 3 possible partition points, and if it contains l such positions, at least $\lfloor l/2 \rfloor \geq l/3$ of them allow to "find" non-fitting elements (i.e., $w_i \neq w_j^R$ for odd $|i-j|$) by Claim 3, independently of the positions of transition points in other sectors of S_{even}. As the choices in different sectors are independent, the probability that none of the choices gives non-fitting elements is smaller than $(2/3)^{\frac{1}{4}\log n} \to 0$ for $n \to \infty$.

Case 2. The map $\mathcal{M}$ contains less than $\frac{1}{2}\log n$ short sectors.

As there are at most $n/\log^3 n$ long sectors, one can easily show that no transition point appears exactly at the left border or the right border of any long sector with pbb $1-o(1)$. Assuming that no transition point is on the borders of the long sector, we choose a long sector i that belongs to the map. Assume that transition points in the remaining sectors are fixed. Thus, the rightmost or the leftmost transition point in the sector i is the candidate to find non-fitting elements (the remaining transition points in the sector i give segments over 0^* or 1^*). The result holds by the following claim and simple calculations which make use of the fact that the length of the block is $\geq \log^3 n$.

Claim 4. *Let $x \in \{0,1\}^m$, where m is even, $m > 0$ and $x \notin j^*$ for some $j \in \{0,1\}$. Then, (at most) one element of the form xj^{2k} and (at most) one element of the form $j^{2k}x$ is a palindrome, where $k \geq 0$.*

References

1. G. Buntrock, F. Otto, *Growing Context-Sensitive Languages and Church-Rosser Languages*, Information and Computation 141(1), 1998, 1–36.
2. J. Hromkovic, G. Schnitger, *Pushdown Automata and Multicounter Machines, a Comparison of Computation Modes*, ICALP 2003, LNCS 2719, 66-80.
3. E. Dahlhaus, M.K. Warmuth, *Membership for growing context-sensitive grammars is polynomial*, Journal of Computer Systems Sciences, 33(3), 1986, 456–472.
4. T. Jurdzinski, K. Lorys, *Church-Rosser Languages vs. UCFL*, in Proc. of International Colloquium on Automata, Languages and Programming (ICALP), 2002, LNCS 2380, 147–158. (full version: `www.ii.uni.wroc.pl/~tju/FullCRL.pdf`).
5. T. Jurdzinski, *The Boolean Closure of Growing Context-Sensitive Languages*, DLT 2006, LNCS, to appear.
6. J. Kaneps, D. Geidmanis, R. Freivalds, *Tally Languages Accepted by Monte Carlo Pushdown Automata*, RANDOM 1997, LNCS 1269, 187–195.
7. M. Li, P. Vitanyi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag 1993.
8. I.I. Macarie, M. Ogihara, *Properties of Probabilistic Pushdown Automata*, Theor. Comput. Sci. 207(1), 1998, 117-130.
9. R. McNaughton, P. Narendran, F. Otto, *Church-Rosser Thue systems and formal languages*, Journal of the Association Computing Machinery, 35 (1988), 324–344.
10. G. Niemann, *Church-Rosser Languages and Related Classes*, PhD Thesis, Univ. Kassel, 2002.
11. G. Niemann, F. Otto, *The Church-Rosser languages are the deterministic variants of the growing context-sensitive languages*, Inf. Comp., 197(1-2), 2005, 1-21.

Sorting Long Sequences in a Single Hop Radio Network[*]

Marcin Kik

Institute of Mathematics and Computer Science,
Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
kik@im.pwr.wroc.pl

Abstract. We propose an algorithm for merging two sorted sequences of length $k \cdot m$ stored in two sequences of m stations of single-hop single-channel radio network, where each station stores a block of of k consecutive elements. The time and energetic cost of this algorithm are $6m \cdot k + 8m - 4$ and $8k + 4\lceil \log_2(m + 1)\rceil + 6$, respectively. This algorithm can be applied for sorting a sequence of length $N = k \cdot n$ in a network consisting of n stations with memory limited by $\Theta(k)$ words. For $k = \Omega(\lg n)$, the energetic cost of such sorting is $O(k \cdot \lg n)$ and the time is $O(N \lg n)$. Moreover, the constants hidden by the big "Oh" are reasonably small, to make the algorithm attractive for practical applications.

1 Introduction

We consider the following problem: A sequence of length $N = n \cdot k$ is distributed among n stations of a single-hop radio network. Each station stores k elements of the sequence. The length of the sequence significantly exceeds the number of stations (i.e. $k = \Omega(\lg n)$). We want to sort this sequence. The stations are synchronized. Time is divided into *slots*. Within a single time slot a single message can be broadcast. We consider *single-hop* network: Message broadcast by any station can be received by any other station. A single message contains $O(\max\{B, \lg N\})$ bits, where B is the number of bits of a single *key* (i.e. element of the input sequence). (Typically $B = \Theta(\lg(N))$.) For any station, sending or listening in a single time slot requires a unit of *energy*. The main goal is to minimize *energetic cost* of the algorithm, i.e. the maximal energy dissipated by any single station. This prolongs the lifetime of the battery powered stations of the network. We also assume that each station can store $\Theta(k)$ words of $\max\{B, \lceil \lg N\rceil\}$ bits each.

The sorting algorithms proposed in [9] and [4] assume that each station stores only one key and it is not not evident how to adopt them to the case when each station stores k keys. The sorting algorithm in [9] is based on energetically balanced selection [8] and obtains energetic cost $O(\lg n)$. On the other hand, [4] contains description of simple merge-sort with energetic cost $O(\lg^2 n)$ that due to its low constants and simplicity is attractive for practical applications.

There exists an algorithm [6] that sorts n elements in time $O(n)$ with energetic cost of broadcasting $O(1)$: Each station s_i listens the keys broadcast by remaining stations

[*] Supported by the European Union within the 6th Framework Programme under contract 001907 (DELIS).

and computes the rank r of its own key. In the rth slot of the next stage s_i broadcasts its key to s_r. This algorithm can be immediately adopted for sorting $k \cdot n$ keys in time $O(kn)$ with energetic cost of broadcasting $O(k)$. However the energetic cost of listening in this algorithm would be $\Theta(k \cdot n)$.

A comparator network sorting sequences of length n can be directly simulated on a single-hop network of n stations: each comparator is simulated in two consecutive time slots, when two endpoints of the comparator exchange their values. The time of such an algorithm (in single channel network) is two times the number of comparators, and the energetic cost is not greater than two times the depth of the network. (Actually, it is twice the maximal number of comparisons performed by a single station.)

We can transform such algorithm into an algorithm for sorting sequences of size $n \cdot k$ using the following standard method for comparator networks (see [5], chapter 5.3.4, exercise 38): Each of the n elements of the sequence is replaced by a sorted block of k elements and each of the comparisons of two elements is replaced by sorting (actually merging) of the corresponding two groups. The energetic cost and time of such operation for each involved station is $2k$: It has to broadcast all its keys and receive all keys of the other station.

For example, the AKS sorting network [1] can be transformed into (impractical) algorithm sorting sequences of length $k \cdot n$ in time $O(k \cdot n \lg n)$ and with energetic cost $O(k \lg n)$ and the Batcher networks [2] can be transformed into algorithms with time $O(k \cdot n \lg^2 n)$ and energetic cost $O(k \cdot \lg^2 n)$.

In this paper we present a practical algorithm (based on the simple merging algorithm from [4]) that merges two sequences of length $k \cdot n$ stored in two sequences of n stations in time $O(k \cdot n)$ and with energetic cost $O(\max\{k, \lg n\})$. For the case $k > \lg n$, the energetic cost is $O(k)$. This algorithm can be used for merge-sorting in time $O(k \cdot n \lg n)$ and energetic cost $O(k \lg n)$. This is asymptotically equivalent to the algorithm obtained from AKS network, but the constants involved are much lower.

2 Preliminaries

The network consists of n stations. Initially we have a sequence of $n \cdot k$ keys evenly distributed among the stations: Each station s_i stores a (sorted) sequence of k keys in the table $key[s_i][1 \dots k]$. (This table is extended in both directions by $key[s_i][0]$ and $key[s_i][k+1]$.) By *interval of* s_i we mean the interval $[key[s_i][1], key[s_i][k]]$. After sorting, all the elements in $key[s_i][1 \dots k]$ are less than all the elements in $key[s_{i+1}]$ $[1 \dots k]$, for $1 \leq i < n$.

For simplicity of description we assume that all the keys are pairwise distinct. We ignore the cost of internal operations inside single stations and for the sake of readability we do not optimize them.

3 Merging

We start with an algorithm for merging two sorted sequences of length $k \cdot m$ (called *a-sequence* and *b-sequence*) stored in two sequences of stations $\langle a_1, \dots, a_m \rangle$ (i.e.

a-stations) and $\langle b_1, \ldots, b_m \rangle$ (i.e. *b-stations*), respectively. The result will be a sorted sequence of length $2k \cdot m$ stored in the sequence of stations $\langle a_1, \ldots, a_m, b_1, \ldots, b_m \rangle$.

3.1 An Overview of the Algorithm

In the procedure Try-Ranking($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) (defined in the following section) each station a_i computes the ranks of all its keys in the b-sequence unless its interval is split by the interval of some b_j, and each station b_j computes the ranks of all its keys in the a-sequence if its interval contains at least one endpoint of the interval of some a_j. (I.e. Its interval neither splits the interval of any a_j nor is disjoint with all intervals a_j.) All the uncomputed ranks are computed by the symmetrical procedure Try-Ranking($\langle b_1, \ldots, b_m \rangle$, $\langle a_1, \ldots, a_m \rangle$). As soon as the ranks are known, the final position of each key in the merged sequence is also known. We finish by performing simple permutation routing, where the destination of each key is its final position in the merged sequence.

3.2 Technical Details

We repeat some technical definitions from [4]. Let T_m denote a balanced binary tree consisting of the nodes $1, \ldots, m$: If $m = 2^k - 1$, for some integer $k > 0$, then T_m is a complete binary tree. If $m = 2^k - 1 - l$, for some positive integer $l < 2^{k-1}$, then the l rightmost nodes on the last level are missing. Thus the *shape* of T_m is identical to the

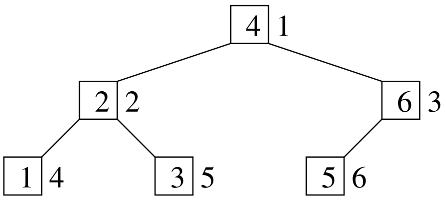


Fig. 1. Tree T_6. Right to the nodes are their *heap-order* indexes.

shape of binary heap containing m elements (see [3]). However, the nodes are placed in T_m in the *in-order* order (i.e. for each node x the nodes in its left subtree are less than x and the nodes in its right subtree are greater than x). By $l(m, x)$ (respectively $r(m, x)$), for $1 \leq x \leq m$, we denote the left (respectively right) child of node x in T_m. (A non-existing child is represented by NIL.) By $p(m, x)$ we denote the index of node x in T_m in *heap-order* ordering, which corresponds to the index of x in an array representation of T_m treated as a heap. (I.e. the *heap-order* index of the root is 1, then the nodes on the second level are indexed from left to right, then on the third level, and so on.) We also assume that $p(m, NIL) = NIL$. An example of T_m for $m = 6$ is given in Figure 1. Note that the height (number of levels) of T_m is $\min\{k : 2^k - 1 \geq m\} = \lceil \lg(m + 1) \rceil$ (where "lg" denotes "$\log_2$"). The algorithm uses several procedures defined below.

```
procedure Init($\langle a_1, \ldots, a_m \rangle$)
begin
    $a_1$ does: $key[a_1][0] \leftarrow -\infty$;
    $a_m$ does: $key[a_m][k+1] \leftarrow +\infty$;
    for time slot $i \leftarrow 1$ to $m-1$ do
        station $a_i$ broadcasts $\langle x \rangle$, where $x = key[a_i][k]$;
        station $a_{i+1}$ listens and does: $key[a_{i+1}][0] \leftarrow x$;
    for time slot $i \leftarrow 1$ to $m-1$ do
        station $a_{i+1}$ broadcasts $\langle x \rangle$, where $x = key[a_{i+1}][1]$;
        station $a_i$ listens and does: $key[a_i][k+1] \leftarrow x$;
end
```

Algorithm 1. Procedure Init

In procedure Init (Algorithm 1) each station is informed about the maximal key stored by its predecessor and the minimal key stored by its successor.

Each station a_i contains additional variables $lPartner[a_i]$, $rPartner[a_i]$, $lRank[a_i]$, and $rRank[a_i]$. $lPartner[a_i]$ and $rPartner[a_i]$ can store either NIL or a triple $\langle x, f, l \rangle$, where x is an index of some b-station b_x and f, l are the endpoints of the interval of b_x. For each a_i, in procedure Find-Partners (Algorithm 3.2), for $k_1 = key[a_i][1]$, we compute in $lPartner[a_i]$ the index and the endpoints of the station b_x such that k_1 is in the interval of b_x. If k_1 is ranked between any two intervals of b_x and b_{x+1}, then $lPartner[a_i] = NIL$ and its final rank in the other sequence (equal $k \cdot x$) is computed in $lRank[a_i]$. (Note that if $x = 0$ then b_x does not exist, and if $x = m$ then b_{x+1} does not exist.) We make analogous computations for $key[a_i][k]$ and variables $rPartner[a_i]$ and $rRank[a_i]$. The computations for each endpoint of a_i are independent. a_i receives only those messages that can influence one of its endpoints. Each time the endpoints of the interval of some b-station are received, the local procedure Update (Algorithm 3.2) is invoked for the endpoint of a_i that can be influenced. Update uses its two initial parameters to update the variables *referenced* by the remaining parameters. Finally, each station a_i uses the values $lPartner[a_i]$, $rPartner[a_i]$, $lRank[a_i]$ and $rRank[a_i]$, to compute $split[a_j]$. The variable $split[a_j]$ becomes true if and only if the interval of some b_j is properly contained in the interval of a_i.

Each station s contains a table $rank[s][1 \ldots k]$. Initially, $rank[a_i][r] = rank[b_i][r] = NIL$, for all $1 \le i \le m$ and $1 \le r \le k$. We want to compute in each $rank[a_i][r]$ the *rank* of $key[a_i][r]$ in b-sequence, and vice versa (i.e. in each $rank[b_i][r]$ the *rank* of $key[b_i][r]$ in a-sequence). By the *rank* of x we mean the number of keys less than x in the other sequence. Procedure Try-Ranking (Algorithm 3.2) computes the ranks in the b-stations that have partners among the a-stations and in each a_j that has $split[a_j] = false$. Procedure Rank (Algorithm 3.3) computes ranks in all the stations, and Merge (Algorithm 3.3) uses the ranks for computing final positions of the keys in the sorted sequence and routes them to their destinations.

3.3 Correctness of Merge

Lemma 1. *For each endpoint e of the interval of each a_i,* Find-Partners($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) *either computes its partner $\langle x, f, l \rangle$ such that $f = key[b_x][0] < e <$*

```
procedure Find-Partners($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$)
begin
    Each station $a_i$ does: begin
        $lTimer[a_i] \leftarrow rTimer[a_i] \leftarrow 1$;
        $lRank[a_i] \leftarrow rRank[a_i] \leftarrow 0$;
        $lParnter[a_i] \leftarrow rPartner[a_i] \leftarrow NIL$;
        $split[a_i] \leftarrow false$;
    end
    for time slot $d \leftarrow 1$ to $m$ do
        let $x$ be such that $p(m, x) = d$; (* $d$ is heap-order index of $x$ *)
        station $b_x$ broadcasts $\langle f, l \rangle$, where $f = key[b_x][1]$ and $l = key[b_x][k]$;
        each station $a_i$ with $d = lTimer[a_i]$ or $d = rTimer[a_i]$ listens and does: begin
            if $d = lTimer[a_i]$ then
                Update($\langle x, f, l \rangle$, $key[a_i][0]$, $lTimer[a_i]$, $lRank[a_i]$, $lPartner[a_i]$);
            if $d = rTimer[a_i]$ then
                Update($\langle x, f, l \rangle$, $key[a_i][k]$, $rTimer[a_i]$, $rRank[a_i]$, $rPartner[a_i]$);
        end
    Each station $a_i$ does: begin
        Let $lP = lParnter[a_i]$, $rP = rPartner[a_i]$, $lR = lRank[a_i]$, and
        $rR = rRank[a_i]$.
        if $(lP = rP = NIL \wedge lR < rR)$ or
        $(lP = \langle x, \ldots \rangle \wedge rP = \langle x', \ldots \rangle \wedge x + 1 < x')$ or
        $(lP = \langle x, \ldots \rangle \wedge rP = NIL \wedge x \cdot k < rR)$ or
        $(lP = NIL \wedge rP = \langle x', \ldots \rangle \wedge lR < (x' - 1) \cdot k)$ then
            (* $a_i$ is split by some $b_j$ *)
            $split[a_i] \leftarrow true$
    end
end
```

Algorithm 2. Procedure Find-Partners

```
procedure Update($\langle x, f, l \rangle$, $key$, $Timer$, $Rank$, $Partner$)
(* $Timer$, $Rank$, and $Partner$ are references to variables *)
begin
    if $f < key < l$ then
        $Partner \leftarrow \langle x, f, l \rangle$;
        $Timer \leftarrow NIL$;
    else if $key < f$ then
        $Timer \leftarrow p(m, l(m, x))$; (* heap-order index of left child of $x$ *)
    else if $l < key$ then
        $Timer \leftarrow p(m, r(m, x))$; (* heap-order index of right child of $x$ *)
        $Rank \leftarrow x \cdot k$; (* $key$ is preceded by at least $x \cdot k$ keys in the other sequence *)
end
```

Algorithm 3. Procedure Update

```
procedure Try-Ranking($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$)
begin
    Init($\langle a_1, \ldots, a_m \rangle$);
    Find-Partners($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$);
    for $i \leftarrow 1$ to $m$ do
        for $r \leftarrow 1$ to $k$ do
            In time slot $2((i-1) \cdot k + r) - 1$:
            $b_i$ broadcasts $\langle v \rangle$, where $v = key[b_i][r]$;
            each $a_j$ with $lPartner[a_j] = \langle i, f, l \rangle$ or $rPartner[a_j] = \langle i, f, l \rangle$ listens and
            does:
            if $split[a_j] = false$ then
                forall $1 \leq s \leq k$ do
                    if $v < key[a_j][s]$ then
                        $rank[a_j][s] \leftarrow (i-1) \cdot k + r$; (* index of $v$ in the $b$-sequence *)
            In time slot $2((i-1) \cdot k + r)$:
            Let $(j, s)$ be the (at most one) pair, such that $lPartner[a_j] = \langle i, f, l \rangle$ or
            $rPartner[a_j] = \langle i, f, l \rangle$ and:

              – $(1 \leq s \leq k \wedge key[a_j][s-1] < v < key[a_j][s])$, or
              – $(s = k + 1 \wedge key[a_j][s-1] < v \leq l < key[a_j][s])$.

            (* I.e. $key[a_j][s]$ is the successor of $v$ in the $a$-sequence and, for $s = k+1$, $b_i$
            is not a partner of $a_{j+1}$. *)
            If such $(j, s)$ exists, then $a_j$ broadcasts $\langle y \rangle$, where $y = (j-1) \cdot k + s - 1$.
            $b_i$ listens and does: begin
                if there was a message $\langle y \rangle$ then
                    $rank[b_i][r] \leftarrow y$; (* index of $key[a_j][s-1]$ in the $a$-sequence *)
            end
    Each $a_i$ does internally: Rank-Unsplit($a_i$);
end
```

Algorithm 4. Procedure Try-Ranking

```
procedure Rank-Unsplit($a_j$)
begin
    if $split[a_j] = false$ then
        if $lPartner[a_j] = rPartner[a_j] = NIL$ then
            for $r \leftarrow 1$ to $k$ do
                $rank[a_j][r] \leftarrow lRank[a_j]$;
        else if $lPartner[a_j] = NIL$ then
            $last \leftarrow \max\{i | key[a_j][i] < f\}$, where $rPartner[a_j] = \langle x, f, l \rangle$;
            for $r \leftarrow 1$ to $last$ do
                $rank[a_j][r] \leftarrow lRank[a_j]$;
end
```

Algorithm 5. Procedure Rank-Unsplit

```
procedure Rank($\langle a_1, \ldots, a_m\rangle$, $\langle b_1, \ldots, b_m\rangle$)
begin
    Each $s \in \{a_1, \ldots, a_m, b_1, \ldots, b_m\}$ does internally: begin
        for $r \leftarrow 1$ to $k$ do $rank[s][r] \leftarrow NIL$;
    end
    Try-Ranking($\langle a_1, \ldots, a_m\rangle$, $\langle b_1, \ldots, b_m\rangle$);
    Try-Ranking($\langle b_1, \ldots, b_m\rangle$, $\langle a_1, \ldots, a_m\rangle$);
end
```

Algorithm 6. Procedure Rank

```
procedure Merge($\langle a_1, \ldots, a_m\rangle$, $\langle b_1, \ldots, b_m\rangle$)
begin
    Rank($\langle a_1, \ldots, a_m\rangle$,$\langle b_1, \ldots, b_m\rangle$);
    Each station $a_i$ does internally:
    for $r \leftarrow 1$ to $k$ do $idx[a_i][r] \leftarrow (i-1)\cdot k + r + rank[a_i][r]$ ;
    Each station $b_i$ does internally:
    for $r \leftarrow 1$ to $k$ do $idx[b_i][r] \leftarrow (i-1)\cdot k + r + rank[b_i][r]$ ;
    (* for $1 \le i \le m$ let $c_i = a_i$ and $c_{m+i} = b_i$ *)
    for time slot $t \leftarrow 1$ to $2m \cdot k$ do
        station $c_i$ with $idx[c_i][r] = t$ broadcasts $\langle k\rangle$, where $k = key[c_i][r]$;
        (* Let $t' = \lfloor (t-1)/k \rfloor + 1$ and $r = t - (t'-1)\cdot k$ *)
        station $c_{t'}$ listens and does: $new[c_{t'}][r] \leftarrow k$;
    Each station $c_i$ does, for $1 \le r \le k$: $key[c_i][r] \leftarrow new[c_i][r]$;
end
```

Algorithm 7. Procedure Merge

$key[b_x][k] = l$ or (if such b_x does not exist) its rank in b-sequence. Variable $split[a_i]$ becomes true if and only if the interval of some b-station is inside the interval of a_i.

Consider arbitrary a_i. Let $e = key[a_i][1]$ (i.e. the left endpoint of a_i). Let $t_1, \ldots, t_r$ be the initial consecutive values (different form NIL) of $lTimer[a_i]$ during the computation. Let $x_1, \ldots, x_r$, be such that $t_j = p(m, x_j)$. Let $\langle f_j, l_j\rangle = \left\langle key[b_{x_j}][1], key[b_{x_j}][k]\right\rangle$. Note that, for each $j < r$, either $e < f_j$ or $l_j < e$ and x_{j+1} is the child of x_j in T_m on the same side that e is to the interval of b_{x_j}. The ordering of intervals of b-stations is the same as the ordering of their indexes. Hence $x_1, \ldots, x_r$ is the path in T_m that should be followed by the ordinary bisection algorithm searching for position of e among the intervals of b-stations. Thus, every time $lRank[a_i]$ and $lTimer[a_i]$ are properly updated by Update. If (after Find-Partners) $lPartner[a_i] \neq NIL$ then its correctness follows from the code of Update. The reasoning for the right endpoint of a_i is the same.

After the variables $lPartner$, $lRank$, $rPartner$ and $rRank$ have been correctly computed in a_i, the correctness of the computation of $split[a_i]$ follows from the observation that some b-station is inside the interval of a_i if and only if a_i has no partners and the ranks of its endpoints are different, or a_i has two partners with indexes more distant than one, or a_i has one partner and the other endpoint of the interval of a_i is not ranked immediately before or immediately after the keys of this partner. □

We have to show that all values in all tables $rank$ are properly computed by Rank. We show that each of these tables is computed by at least one of the two procedures Try-Ranking in Rank. We say that station a_i is *split* in Try-Ranking($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) if it ends up with $split[a_i] = true$. We say that station b_i is *splitting* in Try-Ranking($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) if exists a_j such that $key[a_j][1] < key[b_i][1]$ and $key[b_i][k] < key[a_j][k]$ (i.e. b_i splits a_j). We say that b_i is *avoided* in Try-Ranking($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) if the interval of b_i does not intersect interval of any a_j.

Lemma 2. *Procedure* Try-Ranking($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) *correctly computes all ranks in unsplit a-stations and in unsplitting b-stations that are not avoided. In the remaining stations the tables $rank$ are not modified.*

Let a_j be unsplit. After Find-Partners($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$) there are four possible cases:

Case1: $lPartner[a_j] = rPartner[a_j] = NIL$. In this case interval of a_j is disjoint with all intervals of b-stations and the value $r = lRank[a_j] = rRank[a_j]$ is the rank of each key of a_j in the b-sequence. Procedure Rank-Unsplit(a_j) fills $rank[a_j][1 \ldots k]$ with r.

Case 2: $lPartner[a_j] = \langle x, f, l \rangle$, for some x and $f < l$, and $rPartner[a_j] = NIL$. In the fragment following Find-Partners in Try-Ranking, a_j listens to all keys broadcast in increasing order by b_x during the x-th iteration of the external "for" loop and adjusts all the ranks of its own greater keys. Thus, after the last key of b_x is broadcast, all the ranks in a_j are correct.

Case 3: $lParner[a_j] = NIL$ and $rPartner[a_j] = \langle x, f, l \rangle$, for some x and $f < l$. In this case $lRank[a_j]$ is the proper rank of all keys of a_i less than f. This part of $rank[a_j][1 \ldots k]$ is adjusted in Rank-Unsplit. The remaining ranks are adjusted during the x-th iteration of the external "for" loop after Find-Partners.

Case 4: $lPartner[a_j] = \langle x, f, l \rangle$ and $rPartner[a_j] = \langle x+1, f', l' \rangle$, for some x and $f < l < f' < l'$. The ranks of the keys of a_j that are less than f' are computed during the x-th iteration of the external "for" loop after Find-Partners. Remaining ranks in a_j are computed during the $(x+1)$-st iteration of this loop.

Let b_i be unsplitting and not avoided. Then the the interval of b_i contains the endpoints of all the intervals of a-stations that intersect the interval of b_j. Hence, after Find-Partners, b_i is a partner of all those a-stations and, for each key v of b_i there exists some (unique) a_j with $lPartner[a_j] = \langle i, \ldots \rangle$ or $rPartner[a_j] = \langle i, \ldots \rangle$ that contains successor of v or the last element of a-sequence in the interval of b_i. This station is responsible for answering to the message $\langle v \rangle$ broadcast by b_i. (Note that a_j may be split.)

Let a_j be split. The instructions "if $split[a_j] = false$" in Try-Ranking and Rank-Unsplit prevent modifications of $rank[a_j][1 \ldots k]$.

Let b_i be splitting or avoided. Then b_i is not partner of any a_j and no one answers to its messages broadcast in the ith iteration of the external "for" loop after Find-Partners. Only those answers could have caused modifications of $rank[b_i]$. □

Lemma 3. *If a_i is split in* Try-Ranking($\langle a_1, \ldots, a_m \rangle$, $\langle b_1, \ldots, b_m \rangle$)*, then it is unsplitting and not avoided in* Try-Ranking($\langle b_1, \ldots, b_m \rangle$, $\langle a_1, \ldots, a_m \rangle$).

There is some b_j with its interval properly contained in the interval of a_i. The intervals of $b_1, \dots b_m$ are disjoint. Thus the interval of a_i can not be contained in any one of them (i.e. a_i is unsplitting) and intersects the interval of b_j (i.e. a_i is not avoided). $\square$

Lemma 4. *If b_i is splitting in* Try-Ranking$(\langle a_1, \dots, a_m \rangle, \langle b_1, \dots, b_m \rangle)$*, then it is unsplit in* Try-Ranking$(\langle b_1, \dots, b_m \rangle, \langle a_1, \dots, a_m \rangle)$.

The interval of b_i is properly contained in the interval of some a_j. Thus the procedure Find-Partners$(\langle b_1, \dots, b_m \rangle$, $\langle a_1, \dots, a_m \rangle)$ ends up with $lPartner[b_i] = rPartner[b_i] = \langle j, key[a_j][1], key[a_j][k] \rangle$ and with $split = false$. $\square$

Lemma 5. *If b_i is avoided in* Try-Ranking$(\langle a_1, \dots, a_m \rangle, \langle b_1, \dots, b_m \rangle)$*, then it is unsplit in* Try-Ranking$(\langle b_1, \dots, b_m \rangle, \langle a_1, \dots, a_m \rangle)$.

The interval of b_i is disjoint with each a_j. Thus none interval of a_j can be properly contained in the interval of b_i. $\square$

Lemma 6. *Procedure* Rank *correctly computes all ranks in all stations.*

By Lemmas 2, 3, 4, and 5 all the ranks in all the stations are correctly computed in at least one of Try-Ranking($\langle a_1, \dots, a_m \rangle$, $\langle b_1, \dots, b_m \rangle$) or Try-Ranking($\langle b_1, \dots, b_m \rangle$, $\langle a_1, \dots, a_m \rangle$). (The second Try-Ranking does compute all the ranks missing after the first Try-Ranking and doesn't overwrite any computed ranks with wrong values.) $\square$

Lemma 7. Merge *correctly merges a-sequence with b-sequence.*

Rank computes the rank of each key in the other sequence. Thus the final position of each key in the merged sequence is the sum of its position in its own sequence and its rank in the other sequence. These are exactly the values computed in tables idx. Since all the keys are pairwise distinct, there is exactly one value t in all tables idx, for each $1 \le t \le 2k \cdot m$, and in each iteration of the "for" loop exactly one message is broadcast. (We do not need the reservation phase mentioned in simple routing protocol in [7], since the destinations are *positions in the sequence* – not *indexes of the stations*.) $\square$

3.4 Estimations of Time and Energetic Costs

To make the comparison with other algorithms more fair, we assume that a single message may contain either a single key or a single index of $\lceil \lg(N) \rceil$ bits, where N is total number of keys. Therefore we replace each message $\langle f, l \rangle$ broadcast in Find-Partners by two messages $\langle f \rangle$ and $\langle l \rangle$ broadcast in two consecutive time slots. Let T_M denote the time of Merge. $T_M = T_R + 2m \cdot k$, where T_R is the time of Rank. $T_R = 2T_{TR}$, where T_{TR} is the time of Try-Ranking. $T_{TR} = T_I + T_{FP} + 2m \cdot k$, where T_I is time of Init and T_{FP} is time of Find-Partners. $T_I = 2m - 2$. In Find-Partners, each b_i broadcasts once both endpoints of its interval. Hence, $T_{FP} = 2m$. Thus $T_{TR} = (2m-2)+(2m)+2m \cdot k = 2m \cdot k + 4m - 2$, and $T_R = 4m \cdot k + 8m - 4$, and $T_M = 6m \cdot k + 8m - 4$.

We estimate separately the energetic cost of listening L_M and of sending S_M of Merge. This is more informative in the case when sending requires more energy than listening. However, we assume that the total energetic cost of Merge is $E_M = S_M + L_M$. Thus $S_M = S_R + k$ and $L_M = L_R + k$ (where S_R and L_R are the respective costs of Rank), since in the "for" loop each c_i listens k times and broadcasts each of its keys exactly once. $S_R = S_{TR,a} + S_{TR,b}$ and $L_R = L_{TR,a} + L_{TR,b}$, where $S_{TR,a}$ and $L_{TR,a}$ (respectively, $S_{TR,b}$ and $L_{TR,b}$) are the costs for a-stations (respectively, b-stations) in Try-Ranking. $S_{TR,a} = S_I + 2k$ (where S_I is cost of sending in Init), since some a_i may be obliged to respond to all keys $\langle v\rangle$ broadcast by its both partners. $L_{TR,a} = L_I + L_{FP,a} + 2k$ (where L_I and $L_{FP,a}$ are the listening costs for a-stations in Init and Find-Partners), since each a_i has to listen to all keys $\langle v\rangle$ broadcast by its at most two partners. $S_{TR,b} = S_{FP,b} + k$ (where $S_{FP,b}$ is cost of sending in Find-Partners), since each b_i broadcasts each its key as $\langle v\rangle$. $L_{TR,b} = k + L_{FP,b}$, since each b_i listens to each its message $\langle v\rangle$. $S_{FP,b} = 2$, since each b_i broadcasts its $\langle f, l\rangle$ only once. $L_{FP,a} = 4\lceil\lg(m+1)\rceil$, since each timer, after each updating, becomes the heap-order index of a node on the next level of T_m or NIL. Hence each a_i listens to $\langle f, l\rangle$ at most twice on each level of T_m. $S_{FP,a} = 0$ and $L_{FP,b} = 0$, since a-stations do not broadcast and b-stations do not listen in Find-Partners. It is obvious that $S_I = L_I = 2$. Thus $S_{TR,a} = 2k+2$, $L_{TR,a} = 2k+4\lceil\lg(m+1)\rceil+2$, $S_{TR,b} = k+2$, $L_{TR,b} = k$, $S_R = 3k+4$, $L_R = 3k+4\lceil\lg(m+1)\rceil+2$, $S_M = 4k+4$, and $L_M = 4k+4\lceil\lg(m+1)\rceil+2$. Thus the total energy of Merge is $E_M = 8k + 4\lceil\lg(m+1)\rceil + 6$.

Further Improvements. Since, each message $\langle f, l\rangle$ broadcast in Find-Partners contains the keys that are memorized by all interested a-stations, b-stations do not need repeat sending them in the following "for" loops in Try-Ranking. This reduces the time of Try-Ranking by $2m$ and the sending energy of each b-station and listening energy of each a-station by 2. Thus the energetic cost of Merge is reduced by 4 and its time is reduced by $4m$. We have proven the following theorem:

Theorem 1. *There exists algorithm merging two sorted sequences of length $k \cdot m$, divided into consecutive blocks of size k stored in two sequences of m stations, in time $6m \cdot k + 4m - 4$ with energetic cost $8k + 4\lceil\lg(m+1)\rceil + 2$.*

For comparison consider the Batcher comparator network for merging two sequences of length m (either bitonic or odd-even merge [2]). It contains $\approx \frac{m \lg m}{2}$ comparators. Thus the time of merging a-sequence with b-sequence with the adaptation of this network, as described in Section 1, requires $\approx m \lg m \cdot k$ time slots.

The energetic cost of the algorithm obtained from the Batcher network is $\approx 2k \lg m$, since the depth of the Batcher merging network is $\approx \lg m$.

For example, for $m = 2^{10} = 1024$, the time and energetic cost of our algorithm are $6144 \cdot k + 4092$ and $8k + 46$, respectively. For the adaptation of Batcher networks the time and energetic cost are $\approx 10240 \cdot k$ and $\approx 20 \cdot k$.

4 Sorting

For simplicity, let n be power of two. We can treat any sequence of length $N = n \cdot k$ as n sorted sequences of length k. We merge each pair of consecutive sorted sequences into single sorted sequence. We repeat this operation $\lg n$ times to obtain a single sorted sequence of length n. Let $T_M(m)$ and $E_M(m)$ be the time and energetic cost of merging two sequences of length $m \cdot k$, placed in m stations each. Then the time and energetic cost of our sorting procedure are $T_S(n,k) = \sum_{i=0}^{\lg n-1} n \cdot T_M(2^i)/2^i$ and $E_S(n,k) = \sum_{i=0}^{\lg n-1} E_M(2^i)$. If we apply our merging algorithm, then $T_S(n,k) \leq (3k+2)n \lg n$ and $E_S(n,k) \leq (8k+2)\lg n + \sum_{i=1}^{\lg n} 4i = (8k+2)\lg n + 2(\lg n + 1)(\lg n)$.

On the other hand the energetic cost and time for adaptations of Batcher algorithms are $\approx k \lg^2 n$ and $\approx \frac{1}{2} kn \lg^2 n$.

In merge-sort we can mix our merging algorithm with other merging or sorting algorithms (such as Batcher algorithms) that are more efficient for shorter subsequences. The proper choice depends on both parameters n and k.

Remark. A simulation implemented in Java of the merging procedure described in this paper can be found at [10].

References

1. M. Ajtai, J. Komlós and E. Szemerédi. Sorting in $c \log n$ parallel steps. Combinatorica, Vol. 3, pages 1–19, 1983.
2. K. E. Batcher. Sorting networks and their applications. Proceedings of 32nd AFIPS, pages 307–314, 1968.
3. Th. H. Cormen, Ch. E. Leiserson, R. L. Rivest. Introduction to Algorithms. 1994.
4. M. Kik. Merging and Merge-sort in a Single Hop Radio Network. SOFSEM 2006, LNCS 3831, pp. 341-349, 2006.
5. D. E. Knuth. The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison-Wesley, Reading, Mass. 1973.
6. K. Nakano, S. Olariu. Broadcast-efficient protocols for mobile radio networks with few channels. IEEE Transactions on Parallel and Distributed Systems, 10:1276-1289, 1999.
7. K. Nakano, S. Olariu, A. Y. Zomaya. Energy-Efficient Permutation Routing in Radio Networks. IEEE Transactions on Parallel and Distributed Systems, 12:544-557, 2001.
8. M. Singh and V. K. Prasanna. Optimal Energy Balanced Algorithm for Selection in Single Hop Sensor Network. SNPA ICC, May 2003.
9. M. Singh and V. K. Prasanna. Energy-Optimal and Energy-Balanced Sorting in a Single-Hop Sensor Network. PERCOM, March 2003.
10. Compendium of Large-Scale Optimization Problems. (DELIS, Subproject 3). `http://ru1.cti.gr/delis-sp3/`

Systems of Equations over Finite Semigroups and the #CSP Dichotomy Conjecture

Ondřej Klíma^{1,*}, Benoît Larose^{2,**}, and Pascal Tesson^{3,***}

^{1} Department of Mathematics, Masaryk University
klima@math.muni.cz
^{2} Department of Mathematics and Statistics, Concordia University
larose@mathstat.concordia.ca
^{3} Département d'Informatique et de Génie Logiciel, Université Laval
pascal.tesson@ift.ulaval.ca

Abstract. We study the complexity of counting the number of solutions to a system of equations over a fixed finite semigroup. We show that this problem is always either in FP or #P-complete and describe the borderline precisely. We use these results to convey some intuition about the conjectured dichotomy for the complexity of counting the number of solutions in constraint satisfaction problems.

1 Introduction

Constraint satisfaction problems (or CSPs) are a natural way to formalize a number of computational problems arising from combinatorial optimization, artificial intelligence and database theory. Informally, an instance of CSP consists of a domain, a list of variables and a set of constraints relating the values of the different variables. One then has to decide if the constraints can be simultaneously satisfied. Considerable attention has been given to the case where the constraints are constructed using a finite set of relations Γ and it has been conjectured that for any such Γ the problem CSP(Γ) is either in P or NP-complete [10]. Over the boolean domain Schaefer's classical result [19] states that CSP(Γ) is indeed always in P or NP-complete. More recently, deep results of Bulatov have established a similar dichotomy over the three-element domain [1].

It is similarly believed that the corresponding counting problem #CSP(Γ) is always either tractable (in FP) or #P-complete [3]. This dichotomy is known to hold over the boolean domain [8]. The dichotomy conjectures for CSP and #CSP have been the subject of intense research over the last fifteen years and the algebraic approach uncovered in [13, 12] and extended to the counting problem in [3] has underlied the considerable progress made towards these classification results. It was shown that the tractability of both CSP(Γ) and #CSP(Γ) depends on

* Research supported by the Ministry of Education of the Czech Republic.
** Research supported in part by NSERC and CRM.
*** Research supported in part by NSERC and the Alexander von Humboldt Fellowship programme (while the author was at the University of Tübingen).

the algebraic properties of the set of operations which preserve the relations of Γ. There are now very broad classes of Γ for which the tractability or NP/#P-completeness of CSP(Γ) or #CSP(Γ) can be guaranteed through this algebraic approach. Most notably, Dalmau's recent result on the tractability of CSP(Γ) for any Γ closed under a generalized majority-minority operation provides one of the largest classes of tractable CSPs and, on the hardness side, any Γ which is a core but is not closed under a *Taylor term* is such that CSP(Γ) is NP-complete [7]. This approach also underlies the dichotomy results of Bulatov for CSP over the three-element domain and for the list-homomorphism problem [2].

Despite this remarkable progress, a proof of the dichotomy conjecture for CSP still seems a few years away. For #CSP there are good reasons to be more optimistic: the deep results of [3, 6] provide an algebraic criterion for the tractability of #CSP(Γ) which is known to be necessary and is close to being shown as sufficient (personal communication with A. Bulatov). In their seminal paper [3], Bulatov and Dalmau proved that #CSP(Γ) is #P-complete if Γ is not preserved by any Mal'tsev operation and conjectured that #CSP(Γ) is tractable otherwise. That hypothesis was later refuted by Bulatov and Grohe when they classified the complexity of #CSP(Γ) when Γ consists of two equivalence relations [6].

In order to illuminate the current status of this conjecture, we build on the work of Nordh and Jonsson [18] and study the problem #EQN$^*_S$ of counting the number of solutions to a system of equations over a fixed finite semigroup S. Formally, an equation over S is given as $s_1 \dots s_k = t_1 \dots t_\ell$ where each s_i and t_i is either a constant in S or a variable. An instance of #EQN$^*_S$ is a list of such equations and we are interested in counting the number of ways in which values in S can be assigned to variables such that all equations are satisfied.

We show that for any S, the problem #EQN$^*_S$ is either in FP or #P-complete and precisely describe the class of semigroups involved in the classification (Theorem 15). Such a result is to be expected if one believes that a #CSP dichotomy holds and our classification precisely matches the conjectured criterion for tractability of #CSP. Our results also provide simple examples illustrating the delicate nature of the dividing line between hard and easy cases of #CSP.

Systems of equations over semigroups have already served as interesting case studies for the complexity of CSPs. In [14] it is shown that for any set of relations Γ, there exists a finite semigroup S_Γ such that CSP(Γ) is polynomial time equivalent to the problem EQN$^*_{S_\Gamma}$ of testing if a system over S_Γ has a solution and so proving a dichotomy for this class of problems is equivalent to proving the CSP dichotomy conjecture. If we consider only the problem of solving systems over finite monoids, then the problem is either in P or NP-complete and this result led to the identification of a new class of tractable CSPs [9]. Nordh also considered the problem of testing if two systems are equivalent or isomorphic [17].

We review in Section 2 the basics of the algebraic approach to CSPs and discuss the current conjectures about the #CSP dichotomy. In Section 3, we give the relevant semigroup theoretic notions and rely on deep results of [6, 3] to derive a number of hardness results for #EQN$^*_S$. Finally, we prove our main classification result in Section 4. Due to space restrictions, most proofs have been omitted. An extended version of this paper incorporating the results of [18] is in preparation and its preliminary version is available from the authors' web pages.

2 Universal Algebra and CSPs

Let D be a finite domain and Γ be a finite set of relations over D. The *constraint satisfaction problem over* Γ, denoted CSP(Γ) is the following decision problem. The input consists of a list of variables $x_1, \ldots, x_n$ and constraints that are pairs (S_i, R_i) where R_i is a k_i-ary relation in Γ and S_i, the scope of the constraint, is an ordered list of k_i variables. We ask whether there exists an assignment of values in D to the variables such that every constraint is satisfied. The related counting problem #CSP(Γ) consists of counting the number of such assignments. Throughout the paper, Γ denotes a *constraint language*, i.e. a finite set of relations over some domain D.

The algebraic approach mentioned in our introduction considers the closure properties of Γ. An *operation* f on D is simply a function $f : D^t \to D$. We naturally extend f so that it takes as inputs t k-tuples $\overline{a_1}, \ldots, \overline{a_t}$ of values in D by defining $f(\overline{a_1}, \ldots, \overline{a_t}) = (f(a_{11}, \ldots, a_{t1}), \ldots, f(a_{1k}, \ldots, a_{tk}))$. We say that a k-ary relation R over D is *closed under* f, or that f is a *polymorphism of* R if for any t k-tuples of R, say $\overline{a_1}, \ldots, \overline{a_t}$, we also have $f(\overline{a_1}, \ldots, \overline{a_t}) \in R$. Pictorially,

$$\begin{array}{rccccl}
 & (& a_{11}, & \ldots, & a_{1k} &) \in R \\
 & (& \vdots, & \vdots & \vdots &) \in R \\
 & (& a_{t1}, & \ldots, & a_{tk} &) \in R \\ \cline{2-6}
\Longrightarrow & (& f(a_{11}, \ldots, a_{t1}), & \ldots, & f(a_{1k}, \ldots, a_{tk}) &) \in R
\end{array}$$

In other words, if each of the t rows represents a tuple in R then we can apply f on each of the k columns and again obtain a tuple in R.

By extension we say that Γ is closed under f or that f is a polymorphism of Γ if every relation of Γ is closed under f, and denote as Pol(Γ) the set of all such finitary operations f. The fundamental link to counting CSPs is the following theorem whose counterpart for the decision problem was proved in [12].

Theorem 1 ([3]). *If* Γ_1, Γ_2 *are sets of relations over* D *such that* $\mathrm{Pol}(\Gamma_1) \subseteq \mathrm{Pol}(\Gamma_2)$ *then* #CSP(Γ_2) *is polynomial-time Turing reducible to* #CSP(Γ_1).

A ternary operation m over D is a *Mal'tsev term* if it satisfies the identities $m(x, y, y) = m(y, y, x) = x$. Bulatov and Dalmau showed that if Pol(Γ) contains a Mal'tsev term then CSP(Γ) is tractable [4]. A very broad criterion for #P-completeness of #CSP(Γ) can also be given in terms of these operations.

Theorem 2 ([3]). *If* Γ *is a constraint language such that* Pol(Γ) *contains no Mal'tsev term, then* #CSP(Γ) *is #P-complete.*

It was first conjectured [3] that the presence of a Mal'tsev term in Pol(Γ) was in fact sufficient for the tractability of #CSP(Γ) but later work of Bulatov, Dalmau and Grohe [6, 4] revealed a more complex picture: the algorithm that guarantees the tractability of CSP(Γ) when Pol(Γ) contains a Mal'tsev term cannot quite be adapted to solve #CSP(Γ) efficiently. It can be salvaged in one important special case discussed below.

An *algebra* $\mathbb{D}$ over a domain D is a pair $\langle D; F\rangle$ where F is a set of operations over D, called the *fundamental operations* of $\mathbb{D}$. For an algebra $\mathbb{D}$, we denote as Inv($\mathbb{D}$) the set of relations over D which are preserved by all its fundamental operations. Let $\langle\Gamma\rangle$ denote the set of relations[1] Inv(Pol(Γ)). We say that the constraint language Γ is #-tractable (resp. #P-complete) if #CSP(Γ) is in FP (resp. #P-complete). By extension we say that the algebra $\mathbb{D}$ is #-tractable if every finite $\Lambda \subseteq$ Inv($\mathbb{D}$) is #-tractable and say that $\mathbb{D}$ is #P-complete if there exists a finite subset $\Lambda \subseteq$ Inv($\mathbb{D}$) such that Λ is #P-complete. It follows from Theorem 1 that Γ is #-tractable (resp. #P-complete) iff its associated algebra $\langle D; \mathrm{Pol}(\Gamma)\rangle$ is #-tractable (resp. #P-complete).

It will also be convenient to consider standard algebraic constructions: given an algebra $\mathbb{D}$, we fix some indexing of its fundamental operations, and can then consider *subalgebras*, *homomorphic images* and *products* of algebras (see [16] or [4]). Bulatov and Dalmau have shown that if an algebra is #-tractable then so is every finite algebra obtained from it by these constructions; and conversely, if a power or subalgebra or homomorphic image of an algebra $\mathbb{D}$ is #P-complete then so is $\mathbb{D}$. A *congruence* of an algebra is an equivalence relation on its universe which is invariant under the fundamental operations.

Γ is said to be *uniform* if the following holds: for every binary relation $\theta \in \langle\Gamma\rangle$ such that there exists a subset E of D such that θ is an equivalence relation on E, the blocks of θ all have the same size. Equivalently, Γ is uniform if its associated algebra $\mathbb{D}$ is uniform, i.e. if θ is a congruence of a subalgebra of $\mathbb{D}$ then its blocks all have the same size.

Theorem 3 ([3]). *A uniform algebra admitting a Mal'tsev term is #-tractable.*

This sufficient condition for tractability is not necessary. An algebra is #P-complete if it contains no Mal'tsev term and #-tractable if it is uniform and contains a Mal'tsev term but the dividing line between easy and hard cases of #CSP lies in the small gap between these two criteria. We illustrate this in the next section by giving examples of uniform and non-uniform constraint languages related to systems of equations over Abelian groups.

Bulatov and Grohe considered the complexity of the $\#CSP(\Gamma)$ problem for the special case in which Γ consists of two equivalence relations α, β. For any such α, β, we can construct an integer matrix $M_{\alpha,\beta}$ with rows labeled by the α-classes, columns labeled by β-classes and integer entries given by the size of the intersection of the corresponding α and β classes. Although their result is more general, we cite a weaker theorem that is sufficient for our purposes and really represents the core of their arguments.

[1] Alternatively, $\langle\Gamma\rangle$ is the set of relations expressible through *primitive positive formulas* over Γ and the equality relation (see e.g. [4]).

Theorem 4 ([3]). *If $M_{\alpha\beta}$ is positive and has rank strictly larger than 1, then $\#CSP(\alpha, \beta)$ is #P complete.*

If Γ is a set of relations and α, β are equivalence relations in $\langle \Gamma \rangle$ with $M_{\alpha,\beta}$ positive of rank strictly larger than 1, then $\#\mathrm{CSP}(\Gamma)$ is #P-complete.

It is conjectured that this theorem provides the frontier between the tractable and #P-complete cases of #CSP. More precisely, $\#\mathrm{CSP}(\Gamma)$ should be tractable if $\mathrm{Pol}(\Gamma)$ contains a Mal'tsev term and if for every homomorphic image $\mathbb{B}$ of a subalgebra of a finite power of the algebra associated to Γ, and every pair of congruences α and β of $\mathbb{B}$, the matrix $M_{\alpha,\beta}$ has rank 1 if it is positive. Note that the condition is known to be necessary.

3 Systems of Equations and Dual Algebras

To study the complexity of $\#\mathrm{EQN}^*_S$, we reuse some of the simple but useful observations of [18, 15, 14]. The first concerns the complexity of solving systems over the direct product of two semigroups. One can easily show:

Lemma 5. *Let S and T be finite semigroups such that $\#\mathrm{EQN}^*_T$ is in FP. Then $\#\mathrm{EQN}^*_{S\times T}$, $\#\mathrm{EQN}^*_{S\times S}$ and $\#\mathrm{EQN}^*_S$ are polynomial time Turing equivalent.*

Given a system over S, we can introduce for each $s \in S$ a new variable x_s and the equation $x_s = s$ without affecting the number of solutions to the system. An equation $y_1 y_2 y_3 = z_1 z_2$ can also be replaced by the set of equations $y_1 y_2 = y'$, $y' y_3 = z'$ and $z_1 z_2 = z'$ where y' and z' are new dummy variables, again without affecting the number of solutions. We thus assume that our systems consist of equations of the form $xy = z$, $x = y$ or $x = c$ where x, y, z are variables and $c \in S$ is a constant. Therefore, $\#\mathrm{EQN}^*_S$ can be viewed as a #CSP with domain S and constraint language Γ_S consisting of $|S| + 2$ relations: the $|S|$ singleton unary relations, the equality relation and the ternary relation $\cdot_S = \{(x, y, z) : xy = z\}$. As we explained in Section 2, the complexity of $\#\mathrm{EQN}^*_S$ is completely determined by $\mathrm{Pol}(\Gamma_S)$ and we wish to analyze the structure of that set.

An operation $f : S^k \to S$ *commutes* with the operation $\cdot_S$ of S if for any $s_1, \ldots, s_k, t_1, \ldots, t_k \in S$ we have $f(s_1 t_1, \ldots, s_k t_k) = f(s_1, \ldots, s_k) f(t_1, \ldots, t_k)$. We further say that f is *idempotent* if $f(x, \ldots, x) = x$. For a semigroup S, we denote as $\mathcal{D}(S)$ the *dual algebra* of S, i.e. the algebra over S containing all operations that commute with $\cdot_S$.

Lemma 6 ([15, 18]). *Let Γ_S be the constraint language defined by equations over the semigroup S, then an operation $f : S^k \to S$ is a polymorphism of Γ_S iff f is idempotent and commutes with $\cdot_S$.*

Combining this lemma with Theorems 2 and 3 we get:

Lemma 7.
If $\mathcal{D}(S)$ does not contain a Malt'sev term then $\#\mathrm{EQN}^*_S$ is #P-complete.
If $\mathcal{D}(S)$ is uniform and contains a Malt'sev term then $\#\mathrm{EQN}^*_S$ is tractable.

We shall see that some semigroups fit neither of these criteria but, as a first step, we want to identify the classes of semigroups corresponding to these two cases and this requires some notions of semigroup theory. Recall that an element e of a semigroup S is an *idempotent element* (or simply *an idempotent*) if $e^2 = e$: in a finite semigroup, there exists an integer ω (which has this meaning throughout the paper) such that x^ω is an idempotent for all $x \in S$.

We say that S is a *band* if all its elements are idempotents and say that S is a *left-zero* (resp. *right-zero*) *band* if S satisfies $ab = a$ (resp. $ab = b$). We further say that S is a *rectangular band* if it is the direct product of a right-zero and a left-zero band or, equivalently, if it satisfies $xyz = xz$ and $x^2 = x$.

For a semigroup S, let S^1 denote the monoid obtained from S by adjoining an identity element if no such element exists in S. A semigroup is called *ideal-simple* if for any two elements $a, b \in S$, we have $S^1aS^1 = S^1bS^1$. An equivalent requirement is that for any a, b there exist $x, y \in S^1$ such that $xay = b$. In particular, groups and rectangular bands are ideal-simple semigroups.

A semigroup is said to be *orthodox* if the product of two idempotents of S is itself an idempotent and an ideal-simple semigroup is orthodox iff it is the direct product of a group and a rectangular band [11].

We say that S is an *inflated ideal-simple semigroup* if it consists of an ideal-simple subsemigroup $c(S)$ (the *core* of S) and elements $g_1, \ldots, g_n$ such that for all g_i there exist not necessarily distinct elements $t_1, \ldots, t_n \in c(S)$ satisfying $t_is = g_is$ and $st_i = sg_i$ for all $s \in S$. We say that g_i is a *ghost* of t_i. The terminology of course stresses the fact that the actions defined by left and right multiplication of t_i and g_i are indistinguishable. For an element $t \in c(S)$, we denote as $g(t)$ the set of ghosts of t (including t itself) and call $w(t) = |g(t)|$ the *weight* of t in S. We say that S is a *uniform inflation* of $c(S)$ if each $t \in c(S)$ has the same weight. It can be shown that s is a ghost of t iff $s^{\omega+1} = t$ and, in particular, that two elements of $c(S)$ cannot be ghosts of each other.

Lemma 8. *A finite semigroup S is an inflated ideal-simple orthodox semigroup with only Abelian subgroups iff it satisfies $wxyz = wyxz$ and $xy^\omega z = xz$.*

In the sequel, we denote as $\mathbf{V}$ the class of semigroups which, as in the statement of the lemma. Note that for any $S \in \mathbf{V}$, the core $c(S)$ is always the direct product $A \times L \times R$ of an Abelian group, a left-zero band and a right-zero band. The class $\mathbf{V}$ is tightly connected with dual Malt'sev terms.

Theorem 9. *Let S be a finite semigroup.*
(a) The dual algebra $\mathcal{D}(S)$ contains a Malt'sev term iff S is in $\mathbf{V}$.
(b) If $S \in \mathbf{V}$ then $\mathcal{D}(S)$ is uniform iff S is a uniform inflation of $c(S)$.

We omit the proof. By Lemma 7, we get as an immediate corollary:

Lemma 10. *Let S be the uniform inflation of the direct product of an Abelian group and a rectangular band. Then $\#\mathrm{EQN}^*_S$ is in FP.*

We show in the next section that this lemma does not capture all tractable cases of $\#\mathrm{EQN}^*_S$. First, we establish a general hardness result for solving systems over inflations of Abelian groups.

Theorem 11. *Let S be an inflation of an abelian group A and let $\mathbb{D} = \mathcal{D}(S)$ denote its dual algebra. If S is not a uniform inflation of A then there exist congruences α and β of the algebra $\mathbb{D}^3$ such that the matrix $M_{\alpha\beta}$ is positive of rank strictly greater than 1.*

This theorem and Theorem 4 yield:

Corollary 12. *If S is a non-uniform inflation of an Abelian group then $\#\mathrm{EQN}^*_S$ is #P-complete.*

4 A Classification Result for the Complexity of $\#\mathrm{EQN}^*_S$

Lemma 10 states that $\#\mathrm{EQN}^*_S$ is tractable when S is a uniform inflation of the direct product of an Abelian group and a rectangular band. However, we have not explicitly described polynomial time algorithms for these problems since we invoked the difficult and very general result of [3] on the tractability of uniform Mal'tsev algebras. This is not actually a necessity. First, the fact that the number of solutions to a system of equations over an Abelian group can be computed in polynomial time can be obtained by elementary linear algebra. There is also a very simple polynomial time algorithm to count solutions of a system over any rectangular band $L \times R$ and the proof of Lemma 13 below describes an algorithm for a slightly more general task. These algorithms can be combined to obtain a polynomial time algorithm solving $\#\mathrm{EQN}^*_S$ when S is the direct product of an Abelian group and a rectangular band.

Suppose that we now want to count the number of solutions to a system over an inflation of such a direct product. As noted earlier, we can assume that every equation is of the form $x = y$, $xy = z$ or $z = c$ where c is a constant. We can remove all equations of the form $z = c$ by replacing every occurrence of z by the constant c. In the resulting system, if any solution exists, there exists one in which every variable is set to a value in $c(S)$ because any variable x set to a ghost value s can just as well be set to $s^{\omega+1}$. It is tempting to think that in fact x can be set to any ghost of s but this is not quite the case: if x occurs in an equation of the form $yz = x$ then x can only take values in $c(S)$. We will say that such variables are *regular*. Any solution to the system in which every variable is set to a value in $c(S)$ thus corresponds to a whole set of solutions in which every non-regular variable can be set to any corresponding ghost value. We can formalize these ideas as follows. We say that a solution $\boldsymbol{a} = (a_1, \ldots, a_n)$ is *regular* if all a_i lie in $c(S)$ and define the weight of $\boldsymbol{a}$ as

$$w(\boldsymbol{a}) = \prod_{x_i \text{ non regular}} w(a_i).$$

It is easy to see that the number of solutions to the system is the sum of all $w(\boldsymbol{a})$ where $\boldsymbol{a}$ is a regular solution.

These notions lead to a natural polynomial time algorithm for $\#\mathrm{EQN}^*_S$ when S is a uniform inflation of a direct product of an Abelian group and a rectangular band: since $c(S)$ is uniformly inflated, every regular solution $\boldsymbol{a}$ has the same weight k^t where k is the number of ghosts of any element and t is the number of non-regular variables in the system. Hence $\#\mathrm{EQN}_S^*$ is as hard as computing the number of distinct regular solutions in a system. Finally, it is easy to reduce this task to the problem $\#\mathrm{EQN}_{c(S)}^*$ which is tractable since $c(S)$ is the direct product of an Abelian group and a rectangular band.

The next lemma proves that not all tractable cases of $\#\mathrm{EQN}_S^*$ are captured by Lemma 10.

Lemma 13. *Let S be an inflation of a right-zero band or of a left-zero band. Then $\#\mathrm{EQN}_S^*$ is in FP.*

Proof. We are only interested in summing up the weights of regular solutions. We first identify regular variables and replace any constant by its representative in $c(S)$ (note that we can do this without harm once the equations of the form $x = c$ have been removed). The resulting system can be viewed as a system over the right-zero band $c(S)$ and we want to understand the structure of the set of solutions. Every equation of the form $xy = z$ is in fact equivalent to $y = z$. So if the system has a solution, it is just defining an equivalence relation on the set of variables and constants. Formally, the system partitions the set of variables and constants into classes $Y_{c_1}, \ldots, Y_{c_{|c(S)|}}, X_1, \ldots, X_m$ where the constant c_i lies in Y_{c_i}. We have $\boldsymbol{a}$ a solution, iff all variables in Y_{c_i} are set to c_i and all variables in X_i have the same value a_i. We will abuse notation and denote as $|X_i|$ the number of non-regular variables in the set X_i. Now the weight of $\boldsymbol{a}$ is simply

$$w(\boldsymbol{a}) = \prod |g(c_i)|^{|Y_i|} |g(a_i)|^{|X_i|}.$$

The sum of all these weights is thus

$$\left(\prod |g(c_i)|^{|Y_i|}\right) \left(\prod_{i=1}^{m} \sum_{s \in c(S)} |g(s)|^{|X_i|}\right). \qquad \square$$

Note that if S is a non-uniform inflation of a right-zero band then, by Theorem 9, the dual algebra $\mathcal{D}(S)$ is non-uniform. Thus, the above lemma provides examples of constraint languages Γ such that $\mathrm{Pol}(\Gamma)$ is non-uniform but $\#\mathrm{CSP}(\Gamma)$ is nonetheless tractable.

As we mentioned in Section 2, equivalence relations in $\langle \Gamma \rangle$ play a crucial role in the complexity of $\#\mathrm{CSP}(\Gamma)$. For a semigroup $S \in \mathbf{V}$, there are a number of very natural equivalence relations defined through equations over S. We know that the ideal-simple subsemigroup $c(S)$ can be decomposed as the direct product of an Abelian group A, a right-zero band R and a left-zero band L. Correspondingly, we write an element of this subsemigroup as (a, r, l). Note that since R and L are right and left-zero bands, the multiplication in $c(S)$ is given by $(a_1, r_1, l_1)(a_2, r_2, l_2) = (a_1 a_2, r_2, l_1)$. Let e denote the element $(1_A, r_0, l_0)$ where 1_A denotes the identity element of the group A and r_0, l_0 are arbitrarily chosen elements of respectively R and L. Note that e is an idempotent. Consider the binary relations α_A, α_L and α_R defined as $\alpha_A = \{(x, y) : exe = eye\}$; $\alpha_R = \{(x, y) : ex^\omega = ey^\omega\}$ and $\alpha_L = \{(x, y) : x^\omega e = y^\omega e\}$.

Clearly, all three are equivalence relations and it is not hard to show that they lie in $\langle \Gamma \rangle$. Furthermore ghosts of a same element are equivalent under all three relations. Thus, in each case, an equivalence class is completely determined by its elements in the subsemigroup $c(S) = A \times L \times R$. For an element $x = (a, r, l)$ of $c(S)$, we have $x^\omega e = (1_A, r, l)(1_A, r_0, l_0) = (1_A, r_0, l)$ and so $x, y \in c(S)$ are α_L-equivalent iff their L coordinate is the same. Similarly, x, y are α_R-equivalent iff their R coordinate is the same and are α_A equivalent iff they agree on their group coordinate. The intersection of these three equivalence relations is the equivalence relation $\{(x, y) : x^{\omega+1} = y^{\omega+1}\}$ which equates two elements which are ghosts of the same element of $c(S)$.

Consider the two equivalence relations $\alpha = \alpha_R$ and $\beta = \alpha_A \cap \alpha_L$ and the corresponding matrix $M_{\alpha\beta}$ (as described before Theorem 4). The entries of this matrix correspond to the cardinality of the intersection of an α and a β class. Each such intersection contains precisely a unique element of $c(S)$ and all its ghosts. If the matrix thus formed has rank greater than 1, we know that $\#\mathrm{EQN}^*_S$ is #P-complete by Theorem 4. Otherwise, a folklore fact about positive integer matrices of rank 1 guarantees that $M_{\alpha,\beta}$ is the product of a row vector ρ and a column vector κ which are both positive integer. This allows us to show:

Lemma 14. *Let S be an inflation of $A \times L \times R$. If $M_{\alpha\beta} = \rho\kappa$ has rank 1, then S is isomorphic to the product of an inflation of R and an inflation of $A \times L$.*

Proof. Let $T = A \times L$. Note that two semigroups $S_1, S_2 \in \mathbf{V}$ are isomorphic iff there is an isomorphism ϕ between the ideal-simple semigroups $c(S_1)$ and $c(S_2)$ such that for all x, the ghost classes of x and $\phi(x)$ have the same size.

The matrix $M_{\alpha,\beta}$ has dimension $|R| \times |T|$ and we view rows and columns as being labeled with elements r of R and elements t of T respectively. In S, the number of ghosts of the regular element (r, t) is given by the (r, t) entry of the matrix which is $\rho[r]\kappa[t]$.

Consider the inflation R' of R in which the element r has $\rho[r]$ ghosts and similarly let T' be the inflation of T specified by the column vector κ. It is easy to verify that $R' \times T'$ is indeed isomorphic to S since the number of ghosts of the regular element (r, t) in $R' \times T'$ is the product $\rho[r]\kappa[t]$ of the number of ghosts of r in R' and the number of ghosts of t in T'. □

We can now assemble the pieces of the puzzle to obtain our classification theorem for the complexity of $\#\mathrm{EQN}^*_S$.

Theorem 15. *If S is the direct product of a uniformly inflated Abelian group, an inflated left-zero band and an inflated right-zero band then $\#\mathrm{EQN}^*_S$ is tractable. Otherwise, $\#\mathrm{EQN}^*_S$ is #P-complete.*

Proof. The upper bound for semigroups which are the direct product of a uniformly inflated Abelian group, an inflated left-zero band and an inflated right-zero band follows from Lemmas 10 and 13 as well as Lemma 5 on solving systems over a direct product of semigroups.

For the hardness result, we know by Theorem 9 that any semigroup S outside of $\mathbf{V}$ is such that its dual algebra $\mathcal{D}(S)$ contains no Malt'sev term and Lemma 7 guarantees that $\#\mathrm{EQN}^*_S$ is #P-complete in that case. If S lies in $\mathbf{V}$ then it is an inflation of a direct product $A \times L \times R$ of an Abelian group, a left-zero band and a right-zero band.

Consider the matrix $M_{\alpha,\beta}$ as in Lemma 14: if $M_{\alpha,\beta}$ has rank 2 or more, then $\#\mathrm{EQN}^*_S$ is #P-complete by Theorem 4. If it has rank 1, Lemma 14 allows us to "peel off" an inflated right-zero band out of S. Since the problem $\#\mathrm{EQN}^*_{R'}$ is tractable for any inflation of a right-zero band (Lemma 13), we get by Lemma 5 that the complexity of $\#\mathrm{EQN}^*_S$ is exactly that of $\#\mathrm{EQN}^*_{T'}$ where T' is the inflation of the product of $L \times A$ given by Lemma 14. We can of course repeat the same argument and consider over T' the two equivalence relations $\alpha = \alpha_L$ and $\beta = \alpha_A$ and build the matrix $M_{\alpha,\beta}$. This matrix is positive and if its rank is not 1 then $\#\mathrm{EQN}^*_{T'}$ is again #P-complete by Theorem 4. Otherwise T' is the direct product of an inflation L' of L and an inflation A' of A. Since $\#\mathrm{EQN}^*_{L'}$ is tractable, the problem $\#\mathrm{EQN}^*_S$ is equivalent to the problem $\#\mathrm{EQN}^*_{A'}$. Since we assume that S is not the direct product of a uniformly inflated Abelian group, an inflated left-zero band and an inflated right-zero band it must be that A' is a non-uniform inflation of the Abelian group A. By Corollary 12, $\#\mathrm{EQN}^*_{A'}$ and thus $\#\mathrm{EQN}^*_S$ are $\#P$-complete. □

We argued that $\#\mathrm{EQN}^*_{A'}$ is #P-complete if it is not a uniform inflation of A by using the sophisticated machinery of [4, 6]. An alternative route can also be pursued which we illustrate with the following simple example. Let C_2' be the three-element semigroup consisting of the two-element group C_2 (with the operation written additively) to which we add a ghost for the element 1. In other words, C_2' has elements $\{0, 1, 1'\}$ and the operation is specified by $0+1 = 1+0 = 0+1' = 1'+0 = 1$ and $1+1' = 1'+1 = 1+1 = 1'+1' = 0+0 = 0$.

Since C_2' is a non-uniform inflation of C_2, the #P-completeness of $\#\mathrm{EQN}^*_{C_2'}$ is guaranteed by Corollary 12 but we provide an explicit proof of its hardness by reducing from the #P-complete problem Permanent. Recall that for an $n \times n$ matrix A, the permanent of A is $Perm(A) = \sum_{\sigma \in S_n} \prod_i a_{i\sigma(i)}$. Valiant proved that the problem of computing the permanent of a 0/1 matrix is #P-complete [20].

Theorem 16. *$\#\mathrm{EQN}_{C_2'}$ is #P-complete under Turing reductions.*

Proof. We first prove that the computation of the permanent reduces to the following problem. Given a system $\mathcal{E}$ of equations over the group C_2 with m variables and an integer $0 \le i \le m$, determine the number of solutions to $\mathcal{E}$ that contain i 1's and $m - i$ 0's. We call this problem N_{C_2}.

Let $A = (a_{ij})_{1 \le i,j \le n}$ be the 0/1 matrix whose permanent we wish to compute. We construct a system $\mathcal{E}$ of equations over C_2 with n^2 variables y_{ij} for $1 \le i, j \le n$. There are $2n$ equations in $\mathcal{E}$ corresponding to the n rows and n columns of A.

Specifically, we have for each i an equation $\sum_{j=1}^n a_{ij} y_{ij} = 1$ and for each j an equation $\sum_{i=1}^n a_{ij} y_{ij} = 1$. We claim that $Perm(A)$ is exactly the number of solutions of $\mathcal{E}$ that contain n 1's. Indeed, for each permutation $\sigma \in S_n$ such that $\prod_i a_{i\sigma(i)} = 1$, the assignment to the y_{ij}'s that sets $y_{ij} = 1$ if $j = \sigma(i)$ and $y_{ij} = 0$ otherwise is an assignment with n 1's. Moreover it is a solution of the system because $a_{ij} y_{ij} = 1$ iff $j = \sigma(i)$ and in particular the sum of these products over one row or one column is exactly one. Conversely, every solution to $\mathcal{E}$ with n of the y_{ij} set to 1 must be such that there exists a permutation σ with the property that $y_{ij} = 1$ iff $j = \sigma(i)$ for otherwise at least one row or one column has all its y_{ij}'s set to 0 and an equation is left unsatisfied. Furthermore it must be that $a_{i\sigma(i)} = 1$ for otherwise, again, one of the sums $\sum_{j=1}^{n} a_{ij}y_{ij}$ or $\sum_{i=1}^{n} a_{ij}y_{ij}$ is 0.

Hence there is a one-to-one correspondence between solutions of $\mathcal{E}$ with n 1's and permutations $\sigma \in S_n$ such that $\prod_i a_{i\sigma(i)} = 1$ and so *Perm* reduces to N_{C_2}.

To complete the proof we show a reduction from the problem N_{C_2} to the problem $\#\mathrm{EQN}_{C_2'}$. Let $\mathcal{E}$ be a system of equations over C_2 and suppose for convenience that the n variables in $\mathcal{E}$ are $x_{11}, \ldots, x_{n1}$. We construct a system $\mathcal{E}'$ of equations over the super-semigroup C_2' from the system $\mathcal{E}$ as follows. For each $2 \leq i \leq n$ we introduce $n-1$ new variables x_{ij} for $1 \leq j \leq n$ and add the equations $x_{i1} = x_{i2} = \ldots = x_{in}$. Furthermore we replace any occurrence of a variable y by $y + 0$.

Any solution to $\mathcal{E}$ containing i 1's and $(n-i)$ 0's gives rise to 2^{ni} solutions in $\mathcal{E}'$. Indeed if $x_{i1} = 1$ in the solution to $\mathcal{E}$, then, in $\mathcal{E}'$, x_{i1} can be either 1 or its ghost $1'$ and this is true for each copy x_{ij}. So if N_i is the number of solutions of weight i in $\mathcal{E}$ then the number of solutions in $\mathcal{E}'$ is $\sum 2^{ni}N_i$. Since $N_i \leq \binom{n}{i} < 2^n$ we know that the ith block of n bits in the sum $\sum 2^{ni}N_i$ is precisely N_i. $\square$

This example shows the subtlety of the dividing line between tractable and intractable cases of #CSP. Here, the constraint language $\Gamma_{C_2'}$ is closed under a Mal'tsev operation but $\Gamma_{C_2'}$ is not uniform. Indeed, one can easily verify that the function $m(x, y, z)$ which is x if $y = z$, z if $x = y$ and $x + y + z$ otherwise is a Mal'tsev polymorphism but the equivalence relation $0 + x = 0 + y$ has two equivalence classes $\{0\}$ and $\{1, 1'\}$ of different sizes. Thus, Mal'tsev polymorphisms do not characterize tractable cases of #CSP, even over domains of size three[2].

It is possible to generalize the argument of Theorem 16 to show that any non-uniform inflation of an Abelian group of prime-power order leads to a #P-complete problem. The proof is a tedious case analysis and involves tricks reminiscent of the *thickening* of [6]. With this result in hand, one can now get the #P-completeness result for $\#\mathrm{EQN}^*_{A'}$ when A' is the non-uniform inflation of an Abelian group. We re-apply the reasoning of Theorem 15 to successively peel off from A' uniform inflations of Abelian groups of a given prime power order and thus progressively factor out the tractable components of the $\#\mathrm{EQN}^*_{A'}$ problem. One of three things must happen: in the first case, this process decomposes A' as the direct product of uniform inflation of Abelian groups in which case the counting problem is tractable. In the second case, we hit a positive matrix $M_{\alpha,\beta}$ of rank at least 2, in which case the problem is #P-complete by Theorem 4. In the last case we isolate in A' a non-uniform inflation of an Abelian group of prime power order and the problem is #P-complete.

Acknowledgements. We are grateful to Victor Dalmau and Andrei Bulatov for useful discussions and to the MFCS referees for their many comments.

[2] This contradicts a theorem of [3], retracted in the extended version [5].

References

1. A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proc. of 43rd Foundations of Comp. Sci. (FOCS'02)*, pages 649–658, 2002.
2. A. Bulatov. Tractable conservative constraint satisfaction problems. In *18th IEEE Symp. on Logic in Comp. Sci. (LICS 2003)*, pages 321–331, 2003.
3. A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *44th IEEE Symp. on Foundations of Comp. Sci. (FOCS'03)*, pages 562–571, 2003.
4. A. Bulatov and V. Dalmau. A simple algorithm for mal'tsev constraints. *SIAM J. Comput.*, 36(1):16–27, 2006.
5. A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. 2006. To appear.
6. A. Bulatov and M. Grohe. The complexity of partition functions. In *Proc. 31st Int. Coll. on Automata Languages and Programming*, pages 294–306, 2004.
7. A. Bulatov, A. Krokhin, and P. Jeavons. Constraint satisfaction problems and finite algebras. In *Proc. 27th Int. Coll. on Automata, Languages and Programming—ICALP'00*, pages 272–282, 2000.
8. N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996.
9. V. Dalmau, R. Gavaldà, P. Tesson, and D. Thérien. Tractable clones of polynomials over semigroups. In *Principles and Practice of Constraint Programming—CP'05*, pages 196–210, 2005.
10. T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. on Computing*, 28(1):57–104, 1999.
11. J. Howie. *Fundamentals of Semigroup Theory.* Claredon Press, Oxford, 1995.
12. P. Jeavons. On the algebraic structure of combinatorial problems. *Theor. Comput. Sci.*, 200(1-2):185–204, 1998.
13. P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.
14. O. Klíma, P. Tesson, and D. Thérien. Dichotomies in the complexity of solving systems of equations over finite semigroups. *Theory of Computing Systems*, 2006.
15. B. Larose and L. Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *Internat. J. Algebra and Computation*, 2006. To appear, 17 pages.
16. R. McKenzie, G. McNulty, and W. Taylor. *Algebras, Lattices and Varieties.* Wadsworth and Brooks/Cole, 1987.
17. G. Nordh. The complexity of equivalence and isomorphism of systems of equations over finite groups. In *Math. Found. Comp. Sci. (MFCS'04)*, pages 380–391, 2004.
18. G. Nordh and P. Jonsson. The complexity of counting solutions to systems of equations over finite semigroups. In *Proc. 10th Conf. Computing and Combinatorics (COCOON'04)*, pages 370–379, 2004.
19. T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10^{th} ACM STOC*, pages 216–226, 1978.
20. L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

Valiant's Model: From Exponential Sums to Exponential Products

Pascal Koiran and Sylvain Perifel

LIP, École Normale Supérieure de Lyon
{Pascal.Koiran, Sylvain.Perifel}@ens-lyon.fr

Abstract. We study the power of big products for computing multivariate polynomials in a Valiant-like framework. More precisely, we define a new class $\mathrm{V\Pi P}^0$ as the set of families of polynomials that are exponential products of easily computable polynomials. We investigate the consequences of the hypothesis that these big products are themselves easily computable. For instance, this hypothesis would imply that the nonuniform versions of P and NP coincide. Our main result relates this hypothesis to Blum, Shub and Smale's algebraic version of P versus NP. Let K be a field of characteristic 0. Roughly speaking, we show that in order to separate P_K from NP_K using a problem from a fairly large class of "simple" problems, one should first be able to show that exponential products are not easily computable. The class of "simple" problems under consideration is the class of NP problems in the structure $(K, +, -, =)$, in which multiplication is not allowed.

1 Introduction

Valiant's model. In the framework of Valiant's theory, which goes back to [17], the objects of interest are families of multivariate polynomials. The complexity of such families can be measured by the size of arithmetic circuits which compute them. Two main complexity classes were introduced: VP, whose elements are families of polynomials computed by arithmetic circuits of polynomial size and polynomially bounded degree, and VNP. A VNP family is obtained from a VP family by a summation of (possibly) exponential size, and a central open question is whether VP and VNP coincide. For a long time, these two classes were almost the only classes studied in Valiant's theory. One exception is the class VQP of polynomials computed by arithmetic circuits of quasi-polynomial size of polynomially bounded degree. More recently, new classes were defined and studied by Malod [12]. Of particular interest for us is his class $\mathrm{VP}^0_{\mathrm{nb}}$. In contrast to VP, arbitrary constants are not allowed, and the degrees of polynomials are not bounded.

In this paper we define a new class, called $\mathrm{V\Pi P}^0$, which is obtained from $\mathrm{VP}^0_{\mathrm{nb}}$ by computing products of (possibly) exponential size. By definition $\mathrm{VP}^0_{\mathrm{nb}}$ is included in $\mathrm{V\Pi P}^0$, and we conjecture that this inclusion is strict. Some support for this conjecture is provided by the following observation: if $\mathrm{VP}^0_{\mathrm{nb}} = \mathrm{V\Pi P}^0$ the polynomial family

$$P_d = \prod_{i=0}^{d-1} (X - i) \tag{1}$$

is easy to compute, i.e., can be computed by a family of arithmetic circuits of size polynomial in $\log d$. However, there is in algebraic complexity theory a fairly old conjecture that this family is hard to compute [7,11]. Even more compelling support for our conjecture is provided by Theorem 1, which shows that the non-uniform versions of P and NP coincide if $\mathrm{VP}^0_{\mathrm{nb}} = \mathrm{V\Pi P}^0$, that is, if big products are computable by polynomial size circuits.

The goal of this paper is not merely to define yet another complexity class. Indeed, as explained below the study of $\mathrm{V\Pi P}^0$ leads to meaningful results about the complexity of *decision* problems. This paper is therefore in the same spirit as [9], where it is shown that certain sequences of integers become easy to compute if certain classes of polynomial families coincide.

Blum-Shub-Smale model. One crucial difference between this second model of algebraic computation and Valiant's model is the focus on decision (rather than evaluation) problems. Precise definitions will be given in section 2. In this introduction we will just recall that there is for each field a version of the classical P versus NP problem. In particular, for the field of complex numbers there is a very natural "$\mathrm{P}_{\mathbb{C}} = \mathrm{NP}_{\mathbb{C}}$?" problem, which has remained open since [3]. In order to separate $\mathrm{P}_{\mathbb{C}}$ from $\mathrm{NP}_{\mathbb{C}}$, it is of course sufficient to exhibit a "well chosen" problem A which belongs to $\mathrm{NP}_{\mathbb{C}}$ but not to $\mathrm{P}_{\mathbb{C}}$. One natural choice would be to try $A = \mathrm{FEAS}_{\mathbb{C}}$, where $\mathrm{FEAS}_{\mathbb{C}}$, the feasibility problem for systems of polynomial equations, is the canonical $\mathrm{NP}_{\mathbb{C}}$-complete problem. One insight from Shub and Smale [16] was that there are much more elementary-looking $\mathrm{NP}_{\mathbb{C}}$ problems that do not seem to belong to $\mathrm{P}_{\mathbb{C}}$. Shub and Smale's candidate is the problem Twenty Questions, which can be defined as follows: given a complex number x and an integer d written in binary, decide whether x is an integer in the set $\{0, 1, \dots, d-1\}$. It is not difficult to see that this problem is in $\mathrm{NP}_{\mathbb{C}}$ (hint: guess the binary decomposition of x). Shub and Smale gave compelling evidence that this problem does not belong to $\mathrm{P}_{\mathbb{C}}$, but no conclusive proof could be obtained. In hindsight, this lack of definitive results is not surprising. Indeed, to decide whether an input to Twenty Questions should be accepted it suffices to evaluate the polynomial P_d at $X = x$, and to compare the result to 0. In order to show that Twenty Questions is not in $\mathrm{P}_{\mathbb{C}}$, one must therefore show that the family P_d is hard to compute. As explained above, this is a fairly longstanding open problem[1] which actually predates [16].

In this paper we investigate the following question: are there other examples of "simple" problems which might be used to separate $\mathrm{NP}_{\mathbb{C}}$ from $\mathrm{P}_{\mathbb{C}}$? The class of "simple" problems that we have in mind is $\mathrm{NP}_{(\mathbb{C},+,-,=)}$. This is the class of NP problems over the set of complex numbers endowed with addition, subtraction, and equality tests (there is therefore no multiplication in this structure). It con-

[1] The computation model of [7] and [11] is non-uniform, but Shub and Smale's is uniform. It doesn't seem, however, that adding a uniformity requirement would be of much help in showing that the family P_d is hard to compute.

tains Twenty Questions and many other natural problems (for instance, Subset Sum). Our main result, Theorem 2, is established in section 5: we show that if $\mathrm{VP}^0_{\mathrm{nb}} = \mathrm{V\Pi P}^0$ then $\mathrm{NP}_{(\mathbb{C},+,-,=)}$ is contained in $\mathbb{P}_{(\mathbb{C},+,-,\times,=)}$, the non-uniform version of $\mathrm{P}_{\mathbb{C}}$. Here, the non-uniformity is only due to the fact that (in keeping with the tradition set by Valiant) the classes $\mathrm{VP}^0_{\mathrm{nb}}$ and $\mathrm{V\Pi P}^0$ are non-uniform. One could equally well work with uniform versions of $\mathrm{VP}^0_{\mathrm{nb}}$and $\mathrm{V\Pi P}^0$, and arrive instead at the inclusion $\mathrm{NP}_{(\mathbb{C},+,-,=)} \subseteq \mathrm{P}_{\mathbb{C}}$.

We hope that this paper will help put the focus back from decision problems to evaluation problems. Indeed, we have shown that in order to prove good lower bounds for problems in a fairly large class of decision problems, one must first be able to prove good lower bounds for a related class of evaluation problems. It is a natural question whether the study of evaluation problems can shed light not only on the problem "$\mathrm{NP}_{(\mathbb{C},+,-,=)} \subseteq \mathrm{P}_{\mathbb{C}}$?", but also on the full "$\mathrm{P}_{\mathbb{C}} = \mathrm{NP}_{\mathbb{C}}$?" problem, or on the "$\mathrm{P}_{\mathbb{R}} = \mathrm{NP}_{\mathbb{R}}$?" problem. This question will be investigated in a forthcoming paper.

2 Notations

Our polynomials will be multivariate, and for notational simplicity a tuple of indeterminates will be denoted $\bar{x}$ instead of $(x_1, \ldots, x_{u(n)})$. We will use the Greek letter $\bar{\epsilon}$ to emphasize that we are using a tuple of boolean variables, i.e. $\bar{\epsilon} \in \{0,1\}^{u(n)}$. However, depending on the context $\bar{x}$ will also denote a boolean word when we are dealing with boolean problems.

2.1 Boolean Complexity Classes

We will not offend the reader by defining the boolean classes P and NP. Let us only recall the definitions of their nonuniform versions P/poly and NP/poly. P/poly is defined equivalently in terms of circuits or machines: this is the set of boolean langugages recognized by a family of boolean circuits of polynomial size. Alternatively, this is also the set of languages recognized by a Turing machine working in polynomial time with the help of a polynomial size advice function (hence the name P/poly, see [8]).

NP/poly, the nonuniform version of NP, is the set of languages recognized by polynomial time nondeterministic Turing machine with the help of a polynomial size advice function. Equivalently, it is easily seen to be the nondeterministic counterpart of P/poly, that is to say: $L \in$ NP/poly if and only if there exist $A \in$ P/poly and a polynomial $p(n)$ such that

$$\bar{x} \in L \iff \exists \bar{y} \in \{0,1\}^{p(|\bar{x}|)}.(\bar{x}, \bar{y}) \in A.$$

If A is a language and k a nonnegative integer, $A^{=k}$ denotes the set of words of A of size k.

Another class used in this paper is coRP. It is the set of languages recognized in polynomial time by randomized Turing machines with one-sided error. For more details on boolean complexity, we refer the reader to [13] for instance.

2.2 Algebraic Circuits

In this section we recall the definitions of the non-uniform classes $\mathbb{P}_{(K,+,-,\times,=)}$ and $\mathbb{NP}_{(K,+,-,\times,=)}$, where K is an arbitrary field. These two classes are the non-uniform versions of the classes P_K and NP_K defined by Blum, Shub and Smale [3,2]. Following [14], we will use families of algebraic circuits to recognize languages over K, that is, subsets of $K^\infty = \bigcup_{n\geq 0} K^n$.

An algebraic circuit (understood over $(K,+,-,\times,=)$) is a directed acyclic graph whose vertices, called gates, have indegree 0, 1 or 2. An input gate is a vertex of indegree 0. An output gate is a gate of outdegree 0. We assume that there is only one such gate in the circuit. Gates of indegree 2 are labelled by a symbol from the set $\{+,-,\times\}$. Gates of indegree 1, called test gates, are labelled "$= 0?$". The size of a circuit C, in symbols $|C|$, is the number of vertices of the graph.

A circuit with n input gates computes a function from K^n to K. On input $\bar{u} \in K^n$ the value returned by the circuit is by definition equal to the value of its output gate. The value of a gate is defined in the usual way. Namely, the value of input gate number i is equal to the i-th input u_i. The value of other gates is then defined recursively: it is the sum of the values of its entries for a $+$-gate, their difference for a $-$-gate, their product for a $\times$-gate. The value taken by a test gate is 0 if the value of its entry is $\neq 0$, and 1 otherwise. We assume without loss of generality that the output is a test gate. The value returned by the circuit is therefore 0 or 1.

Finally, the class $\mathbb{P}_{(K,+,-,\times,=)}$ is the set of languages $L \subseteq K^\infty$ such that there exists a tuple $\bar{a} \in K^p$ and a polynomial-size circuit family (C_n) satisfying the following condition: C_n has exactly $n + p$ inputs, and for any $\bar{x} \in K^n$, $\bar{x} \in L \Leftrightarrow C_n(\bar{x}, \bar{a}) = 1$. Note that $\bar{a}$ plays the role of the machine constants of [2,3]. The uniform class P_K of [2,3] can be obtained from $\mathbb{P}_{(K,+,-,\times,=)}$ by adding a uniformity requirement on the family (C_n). In this paper we will stick to non-uniform classes.

Furthermore, $\mathbb{NP}_{(K,+,-,\times,=)}$ is the class of languages L such that there exists a language $A \in \mathbb{P}_{(K,+,-,\times,=)}$ and a polynomial $p(n)$ satisfying

$$\bar{x} \in L \iff \exists \bar{y} \in K^{p(|\bar{x}|)}.(\bar{x}, \bar{y}) \in A.$$

We also define a version $\mathbb{DNP}_{(K,+,-,\times,=)}$ ('D' stands for "digital"), where non-determinism is allowed only on boolean tuples:

$$\bar{x} \in L \iff \exists \bar{y} \in \{0,1\}^{p(|\bar{x}|)}.(\bar{x}, \bar{y}) \in A.$$

We will also need to compute over the structure $(K,+,-,=)$, where multiplication is not allowed. An algebraic circuit over $(K,+,-,=)$ is defined as above, except that there are no $\times$-gate and that there is a new type of gates, called selection gates. A selection gates is of indegree 3. Its value on input (x, y, z) is x if $z = 0$, and y otherwise. The role of these gates is to simulate "if then else" statements. These gates are not needed for the structure $(K,+,-,\times,=)$ since "if then else" statements can be simulated using multiplication (for instance, by the subcircuit $[z = 0?] \times x + (1 - [z = 0?]) \times y)$. The classes $\mathbb{P}_{(K,+,-,=)}$ and $\mathbb{NP}_{(K,+,-,=)}$ are defined in the same way as $\mathbb{P}_{(K,+,-,\times,=)}$ and $\mathbb{NP}_{(K,+,-,\times,=)}$. We could define $\mathbb{DNP}_{(K,+,-,=)}$ as well, but the first author has shown in [10] that $\mathbb{DNP}_{(K,+,-,=)} = \mathbb{NP}_{(K,+,-,=)}$, i.e., only digital nondeterminism is enough over the structure $(K, +, -, =)$.

2.3 Arithmetic Circuits

In Valiant's model, we compute polynomials instead of recognizing languages. A book-length treatment of this topic can be found in [4]. In our framework, which, as explained in the introduction, is not the original one, we require the underlying structure to be a field of characteristic 0, and do not allow arbitrary constants (apart from the constant 1) in our circuits. Hence we compute polynomials $f_n \in \mathbb{Z}[x_1, \ldots, x_{u(n)}]$. Furthermore, we have no restriction on the degree of the polynomials. This formalism was introduced and studied in [12].

An arithmetic circuit is the same as an algebraic circuit over $(K, +, -, \times, =)$, but test gates are not allowed. That is to say we have indeterminates $x_1, \ldots, x_{u(n)}$ as input, $+$, $-$ and $\times$-gates, and we therefore compute polynomials with integer coefficients.

The polynomial computed by an arithmetic circuit is defined in the usual way. Thus a family (C_n) of arithmetic circuits computes a family (f_n) of polynomials, $f_n \in \mathbb{Z}[x_1, \ldots, x_{u(n)}]$. The class $\mathrm{VP}^0_{\mathrm{nb}}$ is the set of families (f_n) of polynomials computed by a family (C_n) of polynomial size arithmetic circuits, i.e., C_n computes f_n and there exists a polynomial $p(n)$ such that $|C_n| \le p(n)$ for all n. We will assume without loss of generality that the number $u(n)$ of variables is bounded by a polynomial function of n.

Arithmetic circuits are at least as powerful as boolean circuits in the sense that one can simulate the latter by the former. Indeed, we can for instance replace $\neg u$ by $1 - u$, $u \wedge v$ by uv, and $u \vee v$ by $u + v - uv$. This proves the following classical lemma.

Lemma 1. *Any boolean circuit C can be simulated by an arithmetic one of size at most $3|C|$, in the sense that on boolean inputs, both circuits output the same value.*

3 Big Products

We introduce here the new class $\mathrm{V\Pi P}^0$, where exponential products are allowed. This is very much inspired by the class VNP, but sums are replaced by products (and, as explained before, constants different from 1 are not allowed, and there is no restriction on the degree).

Definition 1. *The class $\mathrm{V\Pi P}^0$ is the set of families of polynomials $(g_n(\bar{x}))$ such that there exists a family $(f_n(\bar{x}, \bar{y})) \in \mathrm{VP}^0_{\mathrm{nb}}$ satisfying the relation:*

$$g_n(\bar{x}) = \prod_{\bar{\epsilon} \in \{0,1\}^{|\bar{y}|}} f_n(\bar{x}, \bar{\epsilon}).$$

Example 1. The family $(g_n(X))$ defined by $g_n(X) = \prod_{i=0}^{2^n-1}(X-i)$ is in $\mathrm{V\Pi P}^0$. To see this, let $(f_n(X,\bar{\epsilon}))$ be the family

$$f_n(X,\bar{\epsilon}) = X - \sum_{j=1}^{n} \epsilon_j 2^{j-1}.$$

Then $(f_n) \in \mathrm{VP}^0_{\mathrm{nb}}$ and $g_n(X) = \prod_{\bar{\epsilon}\in\{0,1\}^n} f_n(X,\bar{\epsilon})$.

Note that $g_n = P_{2^n}$, where P_{2^n} is defined by (1). This polynomial can therefore be computed by a circuit of size polynomial in n if $\mathrm{VP}^0_{\mathrm{nb}} = \mathrm{V\Pi P}^0$. In fact a more general property holds true: if $\mathrm{VP}^0_{\mathrm{nb}} = \mathrm{V\Pi P}^0$ the family (P_d) is easy to compute. Indeed, once we know how to evaluate efficiently P_d when d is a power of 2, we can also evaluate efficiently for an arbitrary d thanks to the relation $P_{d+2^n}(X) = P_d(X)P_{2^n}(X-d)$. This observation gives some plausibility to the conjecture $\mathrm{VP}^0_{\mathrm{nb}} \neq \mathrm{V\Pi P}^0$. Additional support is provided by Theorem 1 from section 4.

Remark 1. The underlying field is implicit in the notations $\mathrm{VP}^0_{\mathrm{nb}}$ and $\mathrm{V\Pi P}^0$, and should usually be clear from the context. Note that for the question $\mathrm{VP}^0_{\mathrm{nb}} = \mathrm{V\Pi P}^0$, there is no ambiguity at all. Indeed, the equality $\mathrm{VP}^0_{\mathrm{nb}} = \mathrm{V\Pi P}^0$ holds true in a field of characteristic 0 if and only if it holds true in all fields of characteristic 0.

Remark 2. In the spirit of the polynomial hierarchy in boolean complexity theory, one could define a whole hierarchy of new complexity classes by alternating sums and products. The classes $\mathrm{VP}^0_{\mathrm{nb}}$, $\mathrm{VNP}^0_{\mathrm{nb}}$ (also studied by Malod [12]) and $\mathrm{V\Pi P}^0$ would be the first three classes of this hierarchy.

Next we present a criterion which enables to make products over a set more complicated than $\{0,1\}^n$.

Lemma 2. *Let $(f_n(\bar{x},\bar{y}))$ be a* $\mathrm{VP}^0_{\mathrm{nb}}$ *family, and $s(n)$ a function which bounds from above the length of $\bar{y}$, and is itself polynomially bounded (i.e., $s(n) \leq p(n)$ for some polynomial p). Let A be a language in* P/poly. *There exists a family $(g_n(\bar{l},\bar{x}))$ in* $\mathrm{V\Pi P}^0$, *where $|\bar{l}| = s(n) - |\bar{y}|$, such that for any tuple $\bar{x}$ of elements of K and any boolean tuple $\bar{l}$ we have:*

$$g_n(\bar{l},\bar{x}) = \prod_{\bar{\epsilon};\ (\bar{l},\bar{\epsilon})\in A^{=s(n)}} f_n(\bar{x},\bar{\epsilon}).$$

Proof. Since $A \in$ P/poly, there exists a family of polynomial size boolean circuits (C_n) deciding A. By Lemma 1, we can simulate this family of boolean circuits by a family of arithmetic circuits. We obtain a family of polynomials $(c_n(\bar{y},\bar{z}))$ in $\mathrm{VP}^0_{\mathrm{nb}}$ such that for any boolean input $(\bar{l},\bar{\epsilon})$ of size n:

$$c_n(\bar{l},\bar{\epsilon}) = \begin{cases} 1 \text{ if } (\bar{l},\bar{\epsilon}) \in A \\ 0 \text{ otherwise.} \end{cases}$$

The family $(h_n(\bar{x}, \bar{y}, \bar{z}))$ defined by

$$h_n(\bar{x}, \bar{y}, \bar{z}) = c_{s(n)}(\bar{y}, \bar{z}) f_n(\bar{x}, \bar{z}) + 1 - c_{s(n)}(\bar{y}, \bar{z})$$

is therefore in $\mathrm{VP}^0_{\mathrm{nb}}$ and satisfies

$$\prod_{\bar{\epsilon} \in \{0,1\}^{s(n)}} h_n(\bar{x}, \bar{l}, \bar{\epsilon}) = \prod_{\bar{\epsilon};\ (\bar{l}, \bar{\epsilon}) \in A^{=s(n)}} f_n(\bar{x}, \bar{\epsilon}).$$

□

Note that this lemma is already meaningful when $s(n) = |\bar{y}|$, i.e., when $|\bar{l}| = 0$. The more general statement given here will be useful for the proof of our main theorem.

4 Boolean Complexity

In this section we explore the consequences for boolean complexity theory of the assumption that big products are computable by polynomial size circuits. Namely, we prove the following result.

Theorem 1. *If* $\mathrm{V\Pi P}^0 = \mathrm{VP}^0_{\mathrm{nb}}$ *then* P/poly = NP/poly.

Proof. Let $A \in$ NP/poly. Then there exist a language $B \in$ P/poly and a polynomial $p(n)$ such that

$$\bar{x} \in A \iff \exists \bar{y} \in \{0,1\}^{p(|\bar{x}|)}.(\bar{x}, \bar{y}) \in B.$$

Since $B \in$ P/poly, it is decided by a family of polynomial size boolean circuits. These circuits can be simulated by arithmetic ones as in Lemma 1. We obtain a family of polynomials $(f_n(\bar{x}, \bar{y}))$, whose value on a boolean input $(\bar{x}, \bar{y})$ is 0 if $(\bar{x}, \bar{y}) \in B$ and 1 otherwise. This family is in $\mathrm{VP}^0_{\mathrm{nb}}$ because the family of arithmetic circuits has polynomial size.

Now, the products

$$g_n(\bar{x}) = \prod_{\bar{y} \in \{0,1\}^{p(|\bar{x}|)}} f_n(\bar{x}, \bar{y})$$

form a $\mathrm{V\Pi P}^0$ family. On any boolean input $\bar{x}$ we have $g_n(\bar{x}) \in \{0, 1\}$, and $g_n(\bar{x}) = 0$ iff $\exists \bar{y} \in \{0,1\}^{p(|\bar{x}|)}.(f_n(\bar{x}, \bar{y}) = 0)$. In other words,

$$g_n(\bar{x}) = 0 \Leftrightarrow \bar{x} \in A. \tag{2}$$

Under the hypothesis $\mathrm{V\Pi P}^0 = \mathrm{VP}^0_{\mathrm{nb}}$, the family (g_n) is in $\mathrm{VP}^0_{\mathrm{nb}}$. It is therefore computed by polynomial size arithmetic circuits. Deciding whether $\bar{x} \in A$ in nonuniform polynomial time thus amounts to testing in nonuniform polynomial time whether the value of a circuit is zero. It is well known that this can be done in randomized polynomial time coRP by computing modulo random primes (see for instance [15]). The inclusion coRP $\subset$ P/poly [1] concludes the proof. □

It follows from (2) that we can decide any problem in NP by testing an appropriate $\mathrm{V\Pi P}^0$ family for zero. This fact will be used in section 5.3.

5 A Transfer Theorem

We now turn our attention to links with the Blum-Shub-Smale model. The main result of this section, and of the present paper, is the following theorem.

Theorem 2. *If* $\mathrm{V\Pi P}^0 = \mathrm{VP}^0_{\mathrm{nb}}$ *then* $\mathbb{NP}_{(K,+,-,=)} \subseteq \mathbb{P}_{(K,+,-,\times,=)}$.

As in Theorem 1, this connection between $\mathrm{V\Pi P}^0$ and nondeterminism will be obtained by replacing quantifiers by products. However, in $\mathrm{V\Pi P}^0$ the products concern only arithmetic circuits, whereas in $\mathbb{NP}_{(K,+,-,=)}$ the quantifiers concern algebraic circuits (where test gates occur). Therefore, we would like to simulate the computation of an algebraic circuit by an arithmetic one, i.e., to eliminate the test gates. For this purpose, we use boolean circuits as an intermediate step. The latter can indeed be easily simulated by arithmetic circuits by Lemma 1. Doing so requires to deal only with boolean inputs. One part of this problem has already been solved in [10]: boolean nondeterminism already captures $\mathbb{NP}_{(K,+,-,=)}$. It remains to replace the algebraic input $\bar{x} \in K^n$ by a boolean one. This is achieved in the sequel by using mostly techniques which deal with arrangements of hyperplanes. The idea is to replace the algebraic input $\bar{x} \in K^n$ by a point $\bar{q} \in K^n$ of "small" rational coefficients, "close enough" to $\bar{x}$ so that their behaviours will be the same. Now, this rational point can be encoded by boolean tuples, and the whole computation simulated by boolean circuits. "Close enough" means in fact that $\bar{x}$ and $\bar{q}$ belong to the same cell of a suitable arrangement of hyperplanes, i.e., lie on exactly the same hyperplanes of the arrangement. Similar ideas were used in the proofs of the transfer theorems of [5] and [6], which dealt with the structure $(\mathbb{R}, +, -, <)$. Note however that the cells of an arrangement as defined below are not the same as in these two papers. Indeed, since we work in an unordered structure, it doesn't make sense to ask whether a point is "above" or "below" a given hyperplane. The only thing that matters is whether the point lies or not on the hyperplane.

Point location in arrangements of hyperplanes is the main ingredient for finding the rational point $\bar{q}$ on input $\bar{x}$. For a given family of hyperplanes, the goal is to build a circuit which outputs the cell of $\bar{x}$. These notions are explained in section 5.1. In section 5.2 we explain how to find the cell of $\bar{x}$ using $\mathrm{V\Pi P}^0$ tests. Finally, these tools are put together in section 5.3 to recognize $\mathbb{NP}_{(K,+,-,=)}$ problems with the help of $\mathrm{V\Pi P}^0$ tests.

5.1 Arrangement of Hyperplanes

By *hyperplane* (or *affine* hyperplane) of K^n, we mean a surface (of dimension $n-1$) defined by an affine equation $\sum_i \lambda_i x_i = \mu$. We say that k linear hyperplanes of K^n are *independent* if their intersection has dimension exactly $n-k$. In other words, the k hyperplanes are in general position.

An *arrangement of hyperplanes* is merely a finite family of affine hyperplanes $\mathscr{A} = \{H_i;\ i \in I\}$. This enables us to define an equivalence relation:

$$\bar{x} \sim \bar{y} \text{ iff } \forall i.(\bar{x} \in H_i \iff \bar{y} \in H_i).$$

The equivalence classes are called *cells* of the arrangement. In other words, two points are in the same cell if they belong to exactly the same hyperplanes. A cell is therefore of the form

$$(\bigcap_{i\in J} H_i) \setminus (\bigcup_{j\in J'} H_j)$$

for some subsets J and J' of I. One can assume without loss of generality that the hyperplanes $(H_i)_{i\in J}$ are independent. Notice that the cell of $\bar{x} \in K^n$ is characterized by a maximal set (with respect to inclusion) of independent hyperplanes that contain $\bar{x}$. We will use this characterization later for describing the cells of our arrangement. As outlined at the beginning of section 5, on input $\bar{x} \in K^n$ we want to determine its cell, i.e., to return the indices of these independent hyperplanes.

Let $p(n)$ be a fixed polynomial, and $\mathscr{A}_{p,n}$ the set of all hyperplanes in K^n with integer coefficients of absolute value at most $2^{p(n)}$. We call $\mathscr{H}_p$ the family of all the arrangements $\mathscr{A}_{p,n}$ (where n ranges over $\mathbb{N} \setminus \{0\}$). Section 5.2 explains how to build a family of polynomial-size circuits with $\mathrm{V\Pi P}^0$ tests which, on input $\bar{x} \in K^n$, output the cell of $\bar{x}$ in the arrangement $\mathscr{A}_{p,n}$ (this is called "point location" in the arrangement).

5.2 Point Location

The goal of this section is to build an algebraic circuit with $\mathrm{V\Pi P}^0$ tests, which on input $\bar{x} \in K^n$ returns its cell. We first define formally circuits with $\mathrm{V\Pi P}^0$ tests. Then we prove that the point location problem can be solved efficiently using $\mathrm{V\Pi P}^0$ tests.

Definition 2. *A family of algebraic circuits with* $\mathrm{V\Pi P}^0$ *tests is a family* $(f_n(\bar{x})) \in \mathrm{V\Pi P}^0$ *together with a family* (C_n) *of algebraic circuits, where* C_n *is endowed with gates labeled by "*$f_n(\bar{y}) = 0$*?" (the subscript* n *has to be the same for* f_n *and* C_n*). These gates are of indegree* $|\bar{y}|$ *and output 0 if the test fails (i.e.* f_n *evaluated on the inputs of the gate is nonzero), 1 otherwise.*

The class $\mathbb{P}_{(K,+,-,\times,=)}(\mathrm{V\Pi P}^0)$ *is the set of languages recognized by a family of polynomial size algebraic circuits with* $\mathrm{V\Pi P}^0$ *tests.*

By adding some "selection variables", it is not hard to see that in fact any constant number of $\mathrm{V\Pi P}^0$ families can be tested (instead of only one) and still we stay in $\mathbb{P}_{(K,+,-,\times,=)}(\mathrm{V\Pi P}^0)$. For instance, two $\mathrm{V\Pi P}^0$ families will be used in section 5.3: one family to perform a point location task, and the other family to decide a (classical) NP problem. We now explain how to solve the point location problem using $\mathrm{V\Pi P}^0$ tests.

Proposition 1. *Let* $(\mathscr{H}_p)$ *be the family of arrangements of hyperplanes whose coefficients are integers bounded by* $2^{p(n)}$ *in absolute value (this family was defined at the end of section 5.1). There exists a family* (C_n) *of polynomial size algebraic circuits with* $\mathrm{V\Pi P}^0$ *tests that, on input* $\bar{x} \in K^n$*, output the indices of* m *independent hyperplanes that characterize the cell of* $\bar{x}$*.*

Proof. The idea of the algorithm is simple: we maintain a "search space" E which locates $\bar{x}$ as accurately as possible. At the beginning we have no information, and we let $E = K^n$. At each subsequent step, we find (if it exists) the first hyperplane H of our arrangement that refines E, i.e., $\bar{x} \in H$ and $\dim(E \cap H) < \dim E$. At most n steps are therefore enough, and as the description of the cell of $\bar{x}$ we return the indices of the successive hyperplanes met during this process. We will explain below how to find the first hyperplane refining E with the help of $\mathrm{V\Pi P}^0$ tests. Let us first sum up the algorithm:

```
– $E \leftarrow K^n$;
– $L \leftarrow \emptyset$;
– $R \leftarrow \{H \in \mathscr{A} : \bar{x} \in H\}$;
– while $R \neq \emptyset$ do
      . let $H_0$ be the first hyperplane of $R$
      . $L \leftarrow L \cup \{H_0\}$
      . $E \leftarrow E \cap H_0$
      . $R \leftarrow \{H \in \mathscr{A} : \bar{x} \in H$ and $E \cap H \neq E\}$
– return $L$.
```

Note that $E = \bigcap_{H \in L} H$, thus keeping track of L (a list of hyperplanes, or actually of their indices) is enough to determine E.

Finding the first hyperplane refining E is done by binary search, thanks to $\mathrm{V\Pi P}^0$ tests. The list L describing E contains at most n indices, all of size polynomial in n. We store this list in a polynomial number of variables $l_1, \ldots, l_{q(n)}$, representing the boolean encoding of these indices.

At each step, let A be the set of indices of hyperplanes that do not contain E. If f_i is the equation of H_i, the polynomial

$$g(\bar{l}, \bar{x}) = \prod_{i<j \text{ and } i \in A} f_i(\bar{x})$$

vanishes if and only if the first hyperplane refining E has its index smaller than j. By making j vary, we can thus find this hyperplane via binary search in a number of steps which is logarithmic in the number of hyperplanes, i.e., polynomial in n.

We now explain why this product is in $\mathrm{V\Pi P}^0$. With boolean inputs $l_1, \ldots, l_{q(n)}$ and i, we can compute the equation of H_i and test by a simple rank calculation whether H_i has nontrivial trace over E. This is done by a boolean circuit of polynomial size, for instance by Gaussian elimination. Now, Lemma 2 ensures that this product is in $\mathrm{V\Pi P}^0$.

In a polynomial number of $\mathrm{V\Pi P}^0$ tests, we therefore find the first hyperplane refining E. We then proceed with the next step: after at most n steps we have completely characterized the cell of $\bar{x}$. We output the list L of the successive hyperplanes found. This concludes the proof of Proposition 1. □

5.3 Deciding $\mathbb{NP}_{(K,+,-,=)}$ Problems

We are now ready for the main theorem of this section: $\mathbb{NP}_{(K,+,-,=)}$ problems are decided by polynomial size algebraic circuits with $\mathrm{V\Pi P}^0$ tests.

Theorem 3. *Let K be a field of characteristic zero. Then*

$$\mathbb{NP}_{(K,+,-,=)} \subseteq \mathbb{P}_{(K,+,-,\times,=)}(\mathrm{V\Pi P}^0).$$

If big products are computable by arithmetic circuits of polynomial size, one can efficiently simulate $\mathrm{V\Pi P}^0$ tests with algebraic circuits. Theorem 2 therefore follows immediately from Theorem 3.

Proof (of Theorem 3). The outline of the proof is as follows. First we determine the cell of $\bar{x}$. Then we construct in polynomial time a small rational point $\bar{q}$ in the cell. Deciding whether $\bar{q}$ is a positive input is a (classical) NP problem. We have seen in the proof of Theorem 1 that NP problems can be decided by testing a single $\mathrm{V\Pi P}^0$ family for zero. Let us now fill in the details.

Digital nondeterminism. Let $L \in \mathbb{NP}_{(K,+,-,=)}$. By [10, Theorem 2], digital nondeterminism suffices: there exists a language $A \in \mathbb{P}_{(K,+,-,=)}$ and a polynomial $p(n)$ such that

$$\bar{x} \in L \iff \exists \bar{y} \in \{0,1\}^{p(|\bar{x}|)}.(\bar{x},\bar{y}) \in A.$$

Let (C_n) be a family of algebraic circuits of polynomial size $r(n)$ over the structure $(K,+,-,=)$ (i.e. without multiplication gates) that decides A. Notice that our definitions in Valiant's model are constant-free, whereas our algebraic decision circuits (and in particular C_n) may use arbitrary constants. This is not a serious problem: it is enough to consider the constants as new variables (i.e. we pretend that they are part of the input $\bar{x}$), and the circuit is now constant-free. Then our construction leads to a new circuit with the same input variables (and $\mathrm{V\Pi P}^0$ tests). It just remains to plug the original constants in place of the freshly created variables to recognize the original language L. In the remainder of the proof, we therefore assume that the circuits C_n are constant free.

Definition of the arrangement of hyperplanes. Since only addition and subtraction are allowed, on input $(\bar{x},\bar{y})$ every test in C_n is of the form $\sum_{i=1}^{n} \lambda_i x_i = \sum_{i=1}^{p(|\bar{x}|)} \mu_i y_i + \gamma$, where λ_i, μ_i and γ are integers, and are bounded in absolute value by $2^{r(n)}$. Since $y_i \in \{0,1\}$, the right-hand side of the test is bounded in absolute value by $2^{r(n)}(1+p(|\bar{x}|))$. Let $q(n)$ be a polynomial satisfying $2^{q(n)} \geq 2^{r(n)}(1+p(n))$. Consider the family of arrangements $\mathscr{H}_q$ defined in section 5.1: two points $\bar{x}$ and $\bar{x}'$ in the same cell satisfy

$$\forall \bar{y} \in \{0,1\}^{p(|\bar{x}|)} \ [(\bar{x},\bar{y}) \in A \iff (\bar{x}',\bar{y}) \in A].$$

Hence these two points both belong to L, or both belong to its complement.

Finding the cell of $\bar{x}$. We can apply Proposition 1: there is a family of polynomial size algebraic circuits with $\mathrm{V\Pi P}^0$ tests that output the indices of m independent hyperplanes characterizing the cell of $\bar{x}$.

Finding a small rational point in the cell. A point $\bar{q}$ in the cell of $\bar{x}$, of rational coordinates of polynomial size, can be constructed in polynomial time (we omit the description and proof of a boolean algorithm performing this task due to lack of space). As pointed out above, $\bar{x}$ is in L if and only if $\bar{q}$ is in L.

Deciding whether a given rational point belongs to L is a problem in NP. It follows from the proof of Theorem 1 that we can decide whether $\bar{q} \in L$ with one additional $\mathrm{V\Pi P}^0$ test. $\square$

Finally, we thank the anonymous referees for the following remarks.

Remark 3. The $\mathbb{P}_{(K,+,-,\times,=)}$ algorithm of Theorem 3 in fact does not use arithmetic operations (apart from $\mathrm{V\Pi P}^0$ tests of course). Hence the stronger result $\mathbb{NP}_{(K,+,-,=)} \subseteq \mathrm{P}_{(K,=)}(\mathrm{V\Pi P}^0)$ holds. This does not improve Theorem 2, however.

Remark 4. Since $\mathrm{V\Pi P}^0$ can simulate NP (Theorem 1), the inclusion $\mathrm{NP}_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}_{\mathbb{R}_{ovs}}(\mathrm{NP})$ of [6] for $\mathbb{R}_{ovs} = (\mathbb{R}, +, -, \leq)$ implies $\mathrm{NP}_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}_{\mathbb{R}_{ovs}}(\mathrm{V\Pi P}^0)$.

References

1. L. M. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th IEEE symposium on foundations of computer science*, pages 75–83, October 1978.
2. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer-Verlag, 1998.
3. L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.
4. P. Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory.* Number 7 in Algorithms and Computation in Mathematics. Springer, 2000.
5. H. Fournier and P. Koiran. Are lower bounds easier over the reals? In *Proc. 30th ACM Symposium on Theory of Computing*, pages 507–513, 1998.
6. H. Fournier and P. Koiran. Lower bounds are not easier over the reals: Inside PH. In *Proc. ICALP 2000, LNCS 1853*, 2000.
7. J. Heintz and J. Morgenstern. On the intrinsic complexity of elimination theory. *Journal of Complexity*, 9:471–498, 1993.
8. R. M. Karp and R. J. Lipton. Turing machines that take advice. *L'enseignement mathématique*, 28:191–209, 1982.
9. P. Koiran. Valiant's model and the cost of computing integers. *Computational Complexity*, 13:131–146, 2004.
10. P. Koiran. Computing over the reals with addition and order. *Theoretical Computer Science*, 133(1):35–48, 1994.
11. R. J. Lipton. Straight-line complexity and integer factorization. In *Proc. First International Symposium on Algorithmic Number Theory*, volume 877 of *Lecture Notes in Computer Science*, pages 71–79. Springer, 1994.
12. G. Malod. *Polynmes et coefficients.* PhD thesis, Universit Claude Bernard Lyon 1, July 2003.
13. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
14. B. Poizat. *Les petits cailloux.* Aléas, 1995.
15. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, October 1980.
16. M. Shub and S. Smale. On the intractability of Hilbert's Nullstellensatz and an algebraic version of "NP $\neq$ P?". *Duke Math. Journal*, 81(1):47–54, 1995.
17. L. G. Valiant. Completeness classes in algebra. In *Proc. 11th ACM Symposium on Theory of Computing*, pages 249–261, 1979.

A Reachability Algorithm for General Petri Nets Based on Transition Invariants

Alexander E. Kostin

Department of Computer Engineering, Eastern Mediterranean University
Magusa, KKTC, Mersin 10, Turkey
Alexander.kostin@emu.edu.tr

Abstract. A new reachability algorithm for general Petri nets is proposed. Given a Petri net with an initial and a target markings, a so called complemented Petri net is created first that consists of the given Petri net and an additional, complementary transition. Thereby, the reachability task is reduced to calculation and investigation of transition invariants (T-invariants) of the complemented Petri net. The algorithm finds all minimal-support T-invariants of the complemented Petri net and then calculates a finite set of linear combinations of minimal-support T-invariants, in which the complementary transition fires only once. Finally, for each T-invariant with a single firing of the complementary transition, the algorithm tries to create a reachability path from initial to target marking or determines that there is no such path.

Keywords: Petri nets, reachability, transition invariants.

1 Introduction

Petri nets are an important formal paradigm for modeling and analysis of discrete event systems. In such systems, a researcher is often interested to know whether the system can transit from one state into another state. In terms of Petri nets, the answer to this question is obtained as a solution of a reachability problem.

The reachability problem in Petri nets is formulated as follows: for any Petri net *PN*, with an initial marking M^0, and for some other marking M, determine whether the relation $M \in R(PN, M^0)$ is true, where $R(PN, M^0)$ is the reachability set of *PN* for its initial marking M^0 [1]. The decidability of the reachability problem has been proved for a number of restricted classes of Petri nets, and there are efficient algorithms for such classes as acyclic Petri nets, marked graphs, and others [2], [3], [4], [5].

It has been shown that the reachability problem is decidable for general Petri nets as well [6]. In practice, two different techniques are used most often to determine the reachability of a marking in generalized Petri nets. The first technique is based on the creation and investigation of a complete or reduced reachability graph. The main drawback of this approach is a state explosion problem. A closely related technique is the use of *stubborn sets* [8]. Unfortunately, generation of minimal or reduced reachability graphs, as is required by this technique, is known to be an NP-hard problem [10]. If Petri net has no specific properties like a symmetry or reversibility, the corresponding reduced reachability graph will have almost the same size as that of the full reachability graph [9].

The second technique is based on methods of linear algebra. Given a *pure* Petri net (i.e. a Petri net without self-loops), with sets of transitions T and places P, its structure is represented unambiguously by the *incidence matrix*

$$D = [d(t_i, p_j)] = [d_{ij}], \quad i = 1, 2, \dots , m = |T|, \quad j = 1, 2, \dots , n = |P|, \tag{1}$$

where $d(t_i, p_j) = Post(p_j, t_i) - Pre(p_j, t_i)$, Pre and $Post$ are the input and output functions of the Petri net, with $Pre(p, t) = v$ if there is a directed arc from p to t with the weight v, and $Post(p, t) = v$ if there is an arc from t to p with the weight v.

It is known that a *necessary* condition for reachability of marking M from some other marking M^0 is the existence of a nonnegative integer solution of the matrix equation

$$M = M^0 + FD \tag{2}$$

relative to F, where $F = [f_1, f_2, \dots, f_m]$ is a *firing count vector* [12].

Unfortunately, the existence of a nonnegative integer solution of equation (2) is not a sufficient condition for reachability of marking M from M^0 [1]. The second drawback of this method is that the solution of equation (2) does not contain any information about the order of firings. More than that, this can have infinite number of nonnegative integer solutions, some of which work while others fail, and there is a problem to select working firing count vectors [7].

In our paper [11], linear algebra methods were used for reachability analysis of a particular class of place/transition Petri nets having no transition invariants (T-invariants). Algebraically, T-invariants of a Petri net with incidence matrix D are non-negative integer $(1 \times m)$ vectors F such that $FD = 0$ [13].

This paper generalizes the approach described in [11] for arbitrary place/transition nets, including Petri nets with T-invariants. The existence of T-invariants in Petri nets considerably complicates the reachability analysis. In contrast with the scheme in [11], in the generalized scheme the set of T-invariants for investigation is theoretically infinite. Nevertheless, as will be shown in this paper, it is always possible to effectively limit this set without the loss of reachability information and then to use T-invariants from this finite set for performing a reachability analysis.

2 Notation and Basic Statements

We adopt the notation and basic statements from our paper [11]. In particular, the structure of any (pure) Petri net will be represented by its incidence matrix (1), in which rows correspond to transitions and columns correspond to places, as in [1] and [7].

Let M^0 be an initial marking and M be some other marking of given Petri net PN. If we are interested in reachability of M from M^0 then marking M will be called the *target* marking. It is assumed, throughout the paper, that $M^0 \neq M$.

In the paper, all vectors are considered as row vectors. In particular, markings of PN will be expressed as $(1 \times n)$ row vectors, so that we can write

$$M^0 = [m_1^0, m_2^0, \ldots, m_n^0] \quad \text{and} \quad M = [m_1, m_2, \ldots, m_n], \tag{3}$$

where the *i*th entry in M^0 and M denotes the number of tokens in place $p_i \in P$.

If marking M is reachable from M^0 in *PN*, then there exists at least one sequence of markings $\mu = M^0 M^1 \ldots M^r$ with $M^r = M$, and a sequence of firing transitions $\tau = t_{i_1} t_{i_2} \ldots t_{i_r}$, with the two sequences related by the state equation $M^k = M^{k-1} + e[i_k]_m D$, $k = 1, 2, \ldots, r$. Here $e[i_k]_m$ is an $(1 \times m)$ *control vector*, in which $m - 1$ entries are zero and the i_kth entry is one, indicating that a transition t_{i_k} fires at step k. Sequences μ and τ can be combined to obtain a *reachability path* from marking M^0 to M^r:

$$M^0 \xrightarrow{t_{i_1}} M^1 \xrightarrow{t_{i_2}} \ldots \xrightarrow{t_{i_r}} M^r \tag{4}$$

Its determination is the main problem of reachability analysis. With linear algebra methods, this analysis is usually carried out in two stages. The first stage is computing of one or more firing count vectors satisfying equation (2). The second stage is attempting to find reachability paths corresponding to the computed firing count vectors [27]. At the first stage, it is important to limit the number of firing count vectors, without the loss of reachability information. In the proposed approach, this stage is reduced to the computation of T-invariants of so called complemented Petri net.

Definition 1. For any Petri net *PN* with incidence matrix D specified by (1), and initial and target markings M^0 and M represented by vectors (3), there exists a unique *complemented* Petri net PN_c that has the same set of places P as *PN*, the set of transitions $T_c = T \cup \{t_{m+1}\}$, and is described structurally by the incidence matrix

$$D_c = \begin{bmatrix} D \\ \Delta M \end{bmatrix}, \tag{5}$$

where t_{m+1} is an additional, *complementary* transition, and $\Delta M = M^0 - M = [\Delta m_1, \Delta m_2, \ldots, \Delta m_n]$, with $\Delta m_i = m_i^0 - m_i$, $i = 1, 2, \ldots, n$ [11]. ♦

Using the right side of equation (2) with marking M instead of M^0, control vector $e[m+1]_{m+1}$ instead of F and incidence matrix D_c instead of D, one can get $M + e[m+1]_{m+1} D_c = M + \Delta M = M^0$. That is, a single firing of the complementary transition in marking M of PN_c results in marking M^0.

It is known that the reproducibility of a firing sequence in a Petri net indicates the existence of one or more T-invariants [13]. Thus the following statement holds.

Statement 1. Given a Petri net *PN* with the incidence matrix D and an initial marking M^0, a necessary (but generally not sufficient) condition for some other marking $M \neq M^0$ to be reachable from M^0 is the existence of an integer solution of the matrix equation $F_c D_c = 0$ relative to $F_c = [f_1, f_2, \ldots, f_m, f_{m+1}]$, such that $F_c \geq 0$ and $f_{m+1} = 1$. Here D_c is the matrix defined in (5). ♦

In sequel, each T-invariant of the complemented Petri net PN_c having the last entry $f_{m+1} = 1$ will be called a *singular complementary* T-invariant.

The importance of Statement 1 is that the reachability analysis of the *original* Petri net *PN* can be reduced to the computation and investigation of T-invariants of the complemented Petri net PN_c. One advantage of this reduction is the existence of efficient techniques for the calculation of T-invariants [14], [15], [16]. Algorithms for the calculation of T-invariants are implemented in many Petri net software tools such as INA [17], GreatSPN [18], TimeNET [19], and QPN [20], to mention only a few. Even more important benefit of this reduction is that the space for the search of firing sequences, transforming M^0 into M in the given Petri net, can be effectively limited as will be shown in this paper.

It is known that, in any Petri net with T-invariants, there are *minimal-support T-invariants* which can be used as generators of all T-invariants of the given net [1], [13]. Let $\Phi = \{F_1, F_2, \dots, F_s\}$ be the set of minimal-support T-invariants of some Petri net consisting of $m = |T|$ transitions, where $F_i = [f_{i1}, f_{i2}, \dots, f_{im}] \underset{\neq}{>} 0$, and s is the number of minimal-support T-invariants. We use here, for a vector X, a denotation $X \underset{\neq}{>} 0$ if $X \geq 0$ and $x_i \neq 0$ for some ith entry of X. Each $F_i \in \Phi$ specifies a nonempty subset of transitions $\|F_i\| \subseteq T$ such that $t_j \in \|F_i\|$ if and only if $f_{ij} > 0$, with $\|F_i\| \not\subset \|F_k\|$ and $\|F_k\| \not\subset \|F_i\|$ for every pair of distinct indices $i, k = 1, 2, \dots, s$. Here $\|F_i\|$ represents the *minimal support* of T-invariant F_i.

Statement 2. In any Petri net the number of minimal-support T-invariants is finite [11].

Statement 3. For any Petri net *PN*, its complemented net PN_c includes all T-invariants of the original net *PN* [11].

Statement 4. For every reachability path from an initial marking M^0 to a target marking M of a given Petri net *PN*, there exists a T-invariant $F = [f_1, f_2, \dots, f_m, f_{m+1}]$ of the corresponding complemented Petri net PN_c of *PN*, with $f_{m+1} = 1$. That is, F is a singular complementary T-invariant. ♦

Corollary 1. For any Petri net, with given initial and target markings M^0 and M respectively, all existing reachability paths from M^0 to M are the paths that can be created on the set of singular complementary T-invariants. This corollary is a generalization of the corresponding result for T-invariant-less Petri nets in [11]. ♦

Let $\Phi_c = \{F_1, F_2, \dots, F_w\}$ be a set of all minimal-support T-invariants of PN_c, where $F_j = [f_{j1}, f_{j2}, \dots, f_{jm}, f_{j,m+1}] \underset{\neq}{>} 0$, with $j = 1, 2, \dots, w$. Notice that, according to the basic property of a T-invariant, each entry in vector F_j may be only a nonnegative integer [13].

Now, depending on the value of the last entry, the minimal-support T-invariants of set Φ_c can be classified into the following three *disjoint* groups:

$$\{F_j \mid f_{j,m+1} = 0,\ j \in I_w\} \tag{6}$$

$$\{F_j \mid f_{j,m+1} = 1,\ j \in I_w\} \tag{7}$$

$$\{F_j \mid f_{j,m+1} > 1,\ j \in I_w\} \tag{8}$$

where $I_w = \{1, 2, \ldots, w\}$ is the indexing set of Φ_c. According to Statement 2, each of these groups is finite. Depending on the Petri net and its initial and target markings, some or even all these three groups can be empty.

Without the last, (m+1)th entry, T-invariants of group (6), by Statement 3, are minimal-support T-invariants of the original Petri net *PN*. We will call members of group (6) *non-complementary* minimal-support T-invariants of the complemented Petri net PN_c. Group (7) consists of singular complementary T-invariants. Finally, members of group (8) are nonsingular complementary T-invariants in which the complementary transition fires more than once. Together, members of groups (7) and (8) are called minimal-support *complementary* T-invariants of PN_c.

3 Minimal Singular T-Invariants of a Complemented Petri Net

By Corollary 1, the search for all reachability paths from initial marking M^0 to target marking M in a given Petri net can be carried out only on singular T-invariants of the corresponding complemented Petri net. These include, first of all, minimal-support T-invariants of group (7). However, these are not the only singular T-invariants of the complemented Petri net. Indeed, linear combinations of minimal-support T-invariants of groups (6), (7), and (8) can yield additional singular T-invariants. The number of such combinations is infinite in general. However, there exists a finite set of *minimal* singular T-invariants of the complemented Petri net.

Consider a linearly-combined T-invariant

$$F = [f_1, f_2, \ldots, f_m, f_{m+1}] = \sum\nolimits_{j=1}^{w} k_j F_j \tag{9}$$

with rational coefficients k_j, where F_j are minimal-support T-invariants of groups (6), (7) and (8), and w is the number of elements in the three groups. In agreement with Corollary 1, we are looking only for those combined T-invariants F which yield f_{m+1} = 1. Thus, the following constraint must hold for each linear combination F in (9):

$$f_{m+1} = \sum\nolimits_{j=1}^{w} k_j f_{j,m+1} = 1. \tag{10}$$

With $k_j \geq 0$, the product k_jF_j in (9) can be considered as a contribution of firings of transitions of T-invariant F_j to firings of transitions of the combined T-invariant F. On the other hand, a negative coefficient k_j in (9) may be interpreted as a reverse, or backward firing of transitions, corresponding to T-invariant F_j, and this is *not legal* in the normal semantics of Petri nets [21]. Thus, for T-invariants of groups (7) and (8), taking into account (10), their coefficients k_j must be in the following range:

$$0 \leq k_j \leq 1. \tag{11}$$

That is, for groups (7) and (8), in which $f_{j,m+1} \geq 1$, to satisfy (10) the following inequality must hold:

$$\sum k_j \leq 1. \tag{12}$$

However, coefficients k_j for T-invariants of group (6) in (9) may have arbitrary (non-negative) values without affecting the constraint (10). As a particular case, these

T-invariants can be combined in (9) with coefficients $k_j \leq 1$. The case when T-invariants of group (6) can be included into linearly-combined T-invariants (9) with arbitrary large coefficients is considered in Section 6.

The linearly-combined T-invariants (9), with the constraints (10), (11) and (12), are called *minimal singular* T-invariants of the complemented Petri net. As a subset, they include all minimal-support T-invariants of group (7). Minimal singular T-invariants of the complemented Petri net can be found by existing methods of linear algebra [23], [24]. Due to space limitation, we omit the details of the computational procedure.

4 Relation Graph of T-Invariants

In general, each singular T-invariant should be tested for the creation of a reachability path not only *alone*, but also in different linear combinations with non-complementary T-invariants (6), since these T-invariants can "help" the singular T-invariant to become realizable in given initial marking M^0. As will be shown in this section, in general not all non-complementary T-invariants can affect realization of the given singular T-invariant.

Definition 2. Let F be a T-invariant of a Petri net, with the support $\|F\|$. Then $P(F) = \{p_j \mid t_i \in \|F\|, d_{ij} \neq 0\}$ is a set of places of this Petri net *affected* by F when it becomes realizable in some marking. Here, d_{ij} is an element of the incidence matrix(1). ♦

Statement 5. Let F_1 and F_2 be some T-invariants of a Petri net, and let P_1 and P_2 be sets of places affected by F_1 and F_2 respectively. If $P_1 \cap P_2 = \varnothing$, then T-invariants F_1 and F_2 have no *direct* effect on the realizability of each other. ♦

Even if $P_1 \cap P_2 = \varnothing$, T-invariants F_1 and F_2 can *indirectly* affect the realizability of each other through other T-invariants having common affected places with F_1 and F_2. Consider a *relation graph* of T-invariants. Nodes in this graph are T-invariants. Two nodes corresponding to T-invariants F_i and F_j are connected by a non-oriented edge if $P(F_i) \cap P(F_j) \neq \varnothing$, and the corresponding T-invariants F_i and F_j are called *directly connected* T-invariants.

For a Petri net, such a graph generally consists of a number of connected components. A connected component may include complementary and non-complementary T-invariants, or only one type of T-invariants. We say that two T-invariants F_i and F_j can affect realizability of each other if they belong to the same connected component.

The algorithm for determining all connected components of a graph is well known [22]. In our problem, the algorithm will determine a connected component consisting of nodes representing a given singular T-invariant and non-complementary T-invariants. For this purpose, the algorithm will use the incidence matrix of the *original* Petri net and the array of T-invariants.

5 Realization of T-Invariants with Borrowing of Tokens

Let p be a place affected by two T-invariants F_i and F_j in a given Petri net. Assume that, in a given initial marking of the net, F_i is realizable, but F_j can become realizable if place p accumulates r_j tokens during realization of T-invariant F_i. Suppose further that, at some intermediate step during realization of F_i, r_i tokens will be created in place p. If $r_i \geq r_j$ then, by temporary borrowing of r_j tokens in place p, T-invariant F_j becomes realizable and, at the end of its realization, will return the borrowed tokens to place p, so that T-invariant F_i can complete its started realization.

With $r_i < r_j$, T-invariant F_j cannot borrow the necessary number of tokens in place p. However, if T-invariant F_i, after creation of r_i tokens in p at some step of its first realization, can start a new realization before the completion of the first one, then additional r_i tokens will be created in place p, so that this place will now accumulate $2r_i$ tokens. In general, if F_i can start v realizations before the completion of the previous ones, then place p will accumulate vr_i tokens. If, for some v, $vr_i \geq r_j$ then, after borrowing r_j tokens in p, T-invariant F_j becomes realizable. After the completion of its realization, all tokens borrowed by F_j will be returned to place p, and T-invariant F_i can complete all its started realizations.

The possibility of borrowing of tokens among connected T-invariants can be determined with the use of a two-dimensional borrowing matrix G. In this matrix, rows correspond to T-invariants and columns correspond to places of the given Petri net. Formally, $G = [g_{ij}]$, $i = 1, 2, \ldots, s$; $j = 1, 2, \ldots, n$, where s is the number of connected T-invariants and n is the number of places in the net. The elements of matrix G are integers and have the following meaning. If $g_{ij} > 0$ then, for its realization, T-invariant F_i needs to borrow g_{ij} tokens in place p_j affected by some other T-invariants of the considered group. If $g_{ij} < 0$ then T-invariant F_i, at some intermediate step of its *single* realization, creates $|g_{ij}|$ tokens in place p_j. Finally, $g_{ij} = 0$ means that F_i does not affect place p_j.

To create a borrowing matrix, the proposed algorithm will use the incidence matrix of the given original Petri net and a group of connected T-invariants of the corresponding complemented Petri net. Due to a relative simplicity of the underlying procedure and to space limitation, the details of this procedure are omitted.

6 Combining a Singular Complementary T-Invariant with Non-complementary T-Invariants

Denote by F_c a singular T-invariant of some complemented Petri net. It can be a member of group (7) or a minimal T-invariant. If group (6) is not empty, then the following linear combination

$$F = F_c + \sum k_j F_{nc}^j, \tag{13}$$

with coefficients $k_j \geq 0$, is also a valid singular T-invariant, if components of F are nonnegative integers. Here F_{nc}^j is a T-invariant of group (6) connected with F_c.

The expression (13) implies that the singular T-invariant F_c in general should be tested for the creation of a reachability path not only alone, but also in different linear combinations with non-complementary T-invariants (6), since these T-invariants can "help" the non-realizable T-invariant F_c to become realizable in marking M^0.

Without loosing generality, we assume that coefficients k_j in (13) are nonnegative integers. Indeed, if a singular T-invariant F_c is realizable for some non-integer values of coefficients k_j in (13), then it will remain realizable when these coefficient values are replaced by the nearest integer values not less than k_j. The case when $k_j \le 1$ was considered in Section 3. With integer coefficients $k_j > 1$, the product $k_j F_{nc}^j$ in (13) corresponds to a *multiple* realization of T-invariant F_{nc}^j . A multiple realization is a series of k_j sequential or *interleaved* single realizations. Interleaved realizations of a T-invariant, if they are possible in a given marking, can have a different effect on place marking in comparison with sequential realizations. Consider, for example, a simple Petri net consisting of two transitions t_1, t_2 and one place p that is the output place for t_1 and the input place for t_2. This Petri net has a T-invariant $F = [1, 1]$ realizable in any initial marking of p. In particular, with the zero initial marking, place p will never have more than one token if single realizations of F are strictly sequential as in $t_1t_2t_1t_2t_1t_2$. However, if single realizations of F are interleaved, place p can accumulate an arbitrary large number of tokens at some intermediate step.

In general, the number of valid combinations (13) is infinite. This section describes how to limit the values of coefficients k_j in (13) without the loss of reachability information using the concept of structural boundedness of Petri nets.

It is known [1] that a Petri net is *structurally bounded* if and only if there exists a $(1 \times n)$ vector $Y = [y_1, y_2, \ldots, y_n]$ of positive integers, such that

$$D\,Y^T \le 0, \tag{14}$$

where D is the $(m \times n)$-incidence matrix of the Petri net with m transitions and n places.

A Petri net is said to be *structurally unbounded* if and only if there exists a $(1 \times m)$ vector of (nonnegative) integers $X = [x_1, x_2, \ldots, x_m] \underset{\neq}{>} 0$, such that

$$D^T X^T = \Delta M^T \tag{15}$$

for some $\Delta M \underset{\neq}{>} 0$, where m is the number of transitions in the Petri net, and ΔM is a $(1 \times n)$ vector of marking increments as a result of firing of all transitions corresponding to vector X .

In a structurally unbounded Petri net, at least one place is structurally unbounded. A place p_i in such a Petri net is said to be *structurally unbounded* if and only if there exists a $(1 \times m)$ vector $X \underset{\neq}{>} 0$ of nonnegative integers, such that

$$D^T X^T = \Delta M^T \text{ for some } \Delta m_i > 0 \text{ in } \Delta M = (\Delta m_1, \Delta m_2, \ldots, \Delta m_i, \ldots, \Delta m_n) \underset{\neq}{>} 0\,. \tag{16}$$

It is known that, according to Farkas' lemma [1], one of the systems (14) or (15) has solutions. For our problem, we do not need to know all solutions of (14) or (15).

Rather, it is sufficient to find only one, "minimal" solution of (14) or (15). The minimal solutions of (14) or (15) can be found as solutions of integer linear programming (ILP) tasks. For the system (14), the corresponding ILP problem can be formulated as follows:

$$\min \; z=\sum_{i=1}^{n} y_i, \quad \text{sub. to} \quad DY^T \le 0, \; y_i \ge 1, \; i=1, 2,...,n. \tag{17}$$

For the system (15), the corresponding ILP problem is:

$$\min v=\sum_{i=1}^{m} x_i, \text{sub. to} \quad D^T X^T \underset{\neq}{>} 0, \; \sum_{i=1}^{m} x_i \ge 1, \; x_i \ge 0, \; i=1, 2,...,m. \tag{18}$$

Let us consider the case when the subnet corresponding to F_{nc}^{j} is not structurally bounded and describe how to determine coefficient k_j for F_{nc}^{j} in the linear combination (13). If F_{nc}^{j} and F_c belong to different connected components of the relation graph of T-invariants, then F_{nc}^{j} should be ignored at all, by setting $k_j = 0$ in (13). On the other hand, if F_{nc}^{j} and F_c belong to the same connected component of the relation graph of T-invariants, then the subnet corresponding to F_{nc}^{j} has common places with the subnets corresponding to F_c or other non-complementary T-invariants belonging to the same connected component. Thus, F_{nc}^{j} can affect realizability of F_c, directly or indirectly and therefore should be included in (13) with $k_j > 0$.

Suppose, that F_{nc}^{j} has the support $\{t_1, t_2, \ldots, t_l\}$, $l \le m$, and the set of affected places $\{p_1, p_2, \ldots, p_q\}$, $q \le n$, where m and n are the numbers of transitions and places in the original Petri net. Assume that F_c, to become realizable, needs to borrow $n_i > 0$ tokens in each place of set

$$\{p_1, p_2, \ldots, p_h\}, h \le q, \tag{19}$$

in which F_{nc}^{j} can create tokens during its realization. Then, to facilitate the realize-bility of F_c, F_{nc}^{j} should be included in the linear combination (13) with a positive integer coefficient k_j determined by applying the following steps.

1. Try to solve an ILP problem:

$$\min \quad v=\sum_{i=1}^{l} x_i, \quad \text{sub. to} \quad D^T X^T \ge \Delta M^T, \; \sum_{i=1}^{l} x_i \ge 1, \; x_i \ge 0, \tag{20}$$

where $\Delta M = [\Delta m_1, \Delta m_2, .., \Delta m_h, \Delta m_{h+1}, \ldots, \Delta m_q] = [n_1, n_2, \ldots., n_h, 0, \ldots, 0]$ is a vector of the desired numbers of tokens which are expected to be created in places (19) as a result of one or more realizations of F_{nc}^{j}, l is the number of transitions in the support of F_{nc}^{j}, and q is the number of places affected by F_{nc}^{j}. In the matrixmultiplication, only those rows and columns of D are used which correspond to the support of F_{nc}^{j} and places affected by F_{nc}^{j}.

2. If, for the specified vector ΔM, the problem (20) has a solution $X^* = [x_1^*, x_2^*, ..., x_l^*]$, then components of X^* represent the total numbers of firings of respective transitions sufficient to accumulate the desired number of tokens in places of set (19) in a few realizations of F_{nc}^{j}, and $\left\lceil \frac{x_i^*}{f_i^j} \right\rceil$ is the number of realizations of F_{nc}^{j} to get the necessary number of firings of transition t_i, $i = 1, 2, \ldots, l$. In this case,

$$k_j = \max(\left\lceil \frac{x_i^*}{f_i^j} \right\rceil \mid i = 1, 2, ..., l). \tag{21}$$

3. If, on the other hand, the problem (20) has no feasible solution then it means that at least one of places in set (19) p_i is structurally bounded and can not accumulate the desired number of tokens Δm_i in multiple realizations of F_{nc}^{j}. In this case, using (16), determine all structurally unbounded places in set (19).

4. Solve the ILP problem (20) simultaneously for all structurally unbounded places found at the previous step, to obtain a solution vector X^*. That is, in solving (20), vector ΔM should have nonzero entries n_i only for structurally unbounded places. According to Farkas' lemma, this solution always exists. Then coefficient k_j is determined by the use of expression (21).

In case, when the subnet for F_{nc}^{j} is found to be structurally bounded, then the number of tokens in each of its places is bounded. However, this bound generally depends on realizations of other, connected T-invariants and is not known in advance. For such a subnet, coefficient k_j can be evaluated with the use of the borrowing matrix G computed for F_c and all its connected non-complementary T-invariants, including F_{nc}^{j}. Let, in this matrix, c and j be indexes of rows corresponding to F_{nc}^{j} and F_c, respectively. Then it is *sufficient* to include F_{nc}^{j} in (13) with coefficient k_j, computed as

$$k_j = \sum \left\lceil \frac{g_{ci}}{|g_{ji}|} \right\rceil, \tag{22}$$

where g_{ci} and g_{ji} are entries in the borrowing matrix, and the sum is computed for all pairs $g_{ci} > 0$ and $g_{ji} < 0$. Indeed, with this coefficient, the sufficient number of interleaved realizations of F_{nc}^{j} are *allowed* to accumulate the required numbers of tokens in places which are common for F_c and F_{nc}^{j} and in which T-invariant F_c can borrow them during its realization. In general, coefficient k_j calculated as was described can result in a larger number of realizations of T-invariant F_{nc}^{j} than is actually necessary. The reason is that other T-invariants in (13) can also create tokens in (19).

After computing all coefficients k_j in (13), an appropriate method can be applied to determine a reachability path for the combined T-invariant F if such a path exists. For this purpose, known computational procedures can be used [11], [27], [28].

7 An Example

This section illustrates the proposed algorithm by an example. The example was tested with a prototype C program that implemented almost all steps of the algorithm. For solving the related ILP problems the interactive system QS was used [26].

For this example, Fig. 1 shows a Petri net, consisting of m = 10 transitions and n = 9 places (recall that, in the corresponding incidence matrix, rows correspond to transitions). The initial and target markings are M^0 = [2, 0, 0, 0, 0, 0, 0, 0, 0] and M = [2, 0, 0, 0, 0, 0, 0, 0, 1], respectively. To get the complemented Petri net, the algorithm appends a row $\Delta M = M^0 - M$ = [0, 0, 0, 0, 0, 0, 0, 0, -1] to the original incidence matrix.

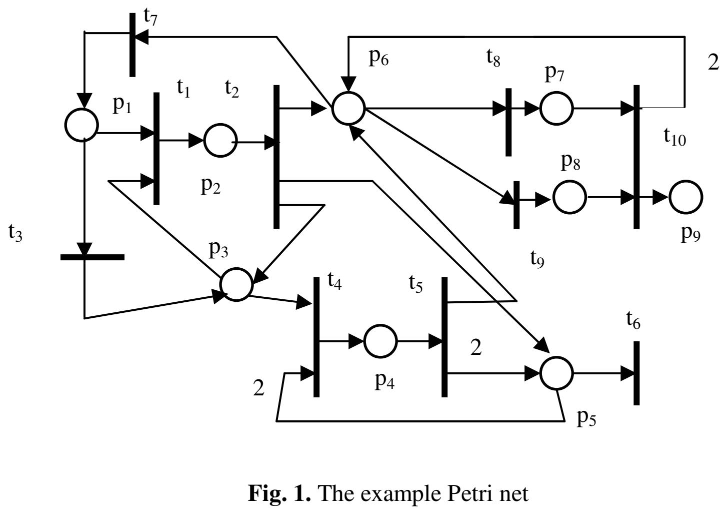


Fig. 1. The example Petri net

Minimal-support T-invariants of the corresponding complemented Petri net are two non-complementary T-invariants F_1 = [0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0] and F_2 = [1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0], and one singular complementary T-invariant F_3 = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1], with the sets of affected places $\{p_1, p_3, p_4, p_5, p_6\}$, $\{p_1, p_2, p_3, p_5, p_6\}$ and $\{p_6, p_7, p_8, p_9\}$, respectively. Thus, all these T-invariants are connected. F_3 can become realizable if it borrows tokens in places affected by F_1 and F_2. All these

T-invariants can become realizable if they borrow tokens in some of their common affected places as the borrowing matrix G for this example shows:

		p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
F_1:	Borrows	-1	0	-1	-1	2	-1	0	0	0
F_2:	Borrows	-1	-1	1	0	-1	-1	0	0	0
F_3:	Borrows	0	0	0	0	0	2	-1	-1	-1

Specifically, F_1 needs to borrow two tokens in place p_5, F_2 needs to borrow one token in place p_3, and F_3 borrows two tokens in place p_6. A token borrowed by F_2 in place p_3 can be produced by F_1 in a single realization so that $k_1 = 1$. On the other hand, F_2 is capable, in a single realization, to lend only one token to F_1, but two tokens are necessary in p_5 for F_1. Therefore, F_1 and F_2 can help each other to become realizable. Together, they are capable to produce two tokens in place p_6 to be borrowed by F_3.

The desired number of tokens in p_5 can be accumulated if the subnet corresponding to F_2 is not structurally bounded. To learn this, the algorithm tries to solve an ILP problem (17) for F_2, in the form in which only variables y_1, y_2, y_3, y_5, and y_6 are taken into consideration:

$$\min z = y_1 + y_2 + y_3 + y_5 + y_6,$$

sub. to: $-y_1 + y_2 - y_3 \le 0$, $-y_2 + y_3 + y_5 + y_6 \le 0$, $-y_5 \le 0$, $y_1 - y_6 \le 0$, $y_1, y_2, y_3, y_5, y_6 \ge 1$.

This ILP problem has no feasible solution. Thus, the subnet corresponding to F_2 is not structurally bounded, so that at least one of its affected places is not structurally bounded. We are interested in accumulating two tokens in p_5, so that $\Delta M = [0, 0, 0, 2, 0]$. Therefore, with $l = 4$ and $q = 5$, the algorithm attempts to solve now an ILP problem (20):

$\min v = x_1 + x_2 + x_6 + x_7$, sub. to: $-x_1 + x_7 \ge 0$, $x_1 - x_2 \ge 0$, $-x_1 + x_2 \ge 0$, $x_2 - x_6 \ge 2$, $x_2 - x_7 \ge 0$, $x_1 + x_2 + x_6 + x_7 \ge 1$.

This ILP problem has the optimal (minimal) solution $X^* = [x_1^*, x_2^*, x_6^*, x_7^*] = [2,2,2,0]$. Now, using (21), the algorithm finds that

$$k_2 = \max\left(\left\lceil \frac{x_i^*}{f_{2i}} \right\rceil \mid i = 1,\ 2,\ 6,\ 7\right) = 2.$$

Since T-invariant F_2 borrows only one token in place p_3 and this token can be created during a single realization of F_1, it is sufficient to have $k_1 = 1$. Thus, the combined T-invariant is $F = F_1 + 2F_2 + F_3 = [2, 2, 1, 1, 1, 2, 3, 1, 1, 1, 1]$. For it, the algorithm creates a reachability path from M^0 to M consisting of 16 nodes, with the sequence of 15 firing transitions $t_3t_1t_2t_7t_1t_2t_4t_5t_6t_6t_8t_9t_{10}t_7t_7$. This is the shortest path although there exist other paths of this length.

8 Conclusion

A new reachability algorithm for general Petri nets is proposed. For a given original Petri net, the reachability task is reduced to the investigation of T-invariants of the complemented Petri net consisting of the original Petri net and an additional, complementary transition. It is shown that, without the loss of reachability information, the algorithm tries to find a reachability path from the initial marking to the target one using a finite number of T-invariants. During the search for reachability paths, the algorithm needs memory for storing only the reachability path being created.

We did not address, in this paper, complexity aspects of the proposed algorithm. Complexity of some problems of Petri nets was investigated in [25]. We can note only that the algorithm will spend most of its time calculating minimal-support T-invariants, solving ILP problems, and trying to find reachability paths for calculated T-invariants.

References

1. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proc. of the IEEE, vol. 77, no. 4 (1989) 541 – 580
2. Ichikawa, A., Hiraishi, K.: A Class of Petri Nets that a Necessary and Sufficient Condition for Reachability is Obtainable. Trans. SICE, vol.24, no. 6 (1988)
3. Kodama, S., Murata, T.: On Necessary and Sufficient Reachability Condition for Some Subclasses of Petri Nets, TR UIC-EECS 88-8, University of Illinois at Chicago, June (1988)
4. Caprotti, O., Ferscha, A., Hong, H.: Reachability Test in Petri Nets by Groebner Bases, TR No. 95-03, Johannes Kepler University, Austria (1995)
5. Kostin, A.E.: The Novel Algorithm for Determining the Reachability in Acyclic Petri Nets. SIGACT News, vol. 28, no. 2 June (1997) 70 - 79
6. 6. Mayr, E.W.: An Algorithm for the General Petri Net Reachability Problem. SIAM Journal of Computing, vol. 13, no. 3 (1984) 441 – 459
7. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall (1981)
8. Varpaaniemi, K.: On the Stubborn Set Method in Reduced State Space Generation, PhD Thesis, Dept. of Computer Science and Engineering, Helsinki University of Technology (1998)
9. Schmidt, K.: Stubborn Sets for Model Checking the EF/AG Fragment of CTL. Fundamenta Informaticae, vol. 43, no. 1 – 4 (2000) 331 – 341
10. Peled, D.: All from One, One from All. LNCS, vol. 697, Springer-Verlag (1993) 409 – 423
11. Kostin, A. E.: Reachability Analysis in T-Invariant-less Petri Nets. IEEE Trans. on Automatic Control, vol. 48, no. 6 (2003) 1019 - 1024
12. Murata, T.: State Equation, Controllability, and Maximal Matchings of Petri Nets. IEEE Transactions on Automatic Control, vol. AC-22, no. 3, (1977) 412 – 416
13. Memmi, G., Roucairol, G.: Linear Algebra in Net Theory. In: Brauer, W. (ed.), Net Theory and Applications, LNCS, vol. 84, Springer-Verlag (1980) 213 – 223
14. Martinez, J., Silva, M.: A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net. In: Girault, C., and Reisig, W. (eds.), Application and Theory of Petri Nets, Springer-Verlag (1982) 301 – 310
15. Alaiwan, H., Toudic, J.-M.: Recherche des semi-flots, des verrous et des trappes dans les reseaux de Petri. Technique et Science Informatique, vol. 4, no. 1 (1985) 103 – 112
16. Anishev, P.A., Bandman, O L.: Algorithms and Programs for the Analysis of Properties of Petri Nets. Preprint no. 762, Academy of Science of USSR, Novosibirsk (1988)

17. Roch, S. Starke, P.H.: INA: Integrated Net Analyzer, Ver. 2.2, Humboldt-Universitat zu Berlin (2001)
18. Chiola, G., Franceschinis, G., Gaeta, R., Ribaudo, M.: GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. Perform. Evaluation, vol. 24, no. 1&2 (1995) 47 – 68
19. German, R., Kelling, C., Zimmerman, A., Hommel, G.: TimeNET: A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets. Perform. Evaluation, vol. 24, no. 1&2 (1995) 69 – 87
20. Bause, F., Kemper, P.: QPN-Tool for the Qualitative and Quantitative Analysis of Queuing Petri Nets. Lecture Notes in Computer Science, vol. 794, Springer-Verlag (1994) 321 – 334
21. Murata, T.: A Private Communication, April 18 (1998)
22. Goodrich, M.T., Tamassia, R.: Algorithm Design: Foundations, Analysis and Internet Examples, John Wiley & Sons (2002)
23. Springer, J.: Exact Solution of General Integer Systems of Linear Equations. ACM Trans. on Mathematical Software, vol. 12, no. 1, March (1986) 51 – 61
24. Howell, J.A.: Exact Solution of Linear Equations Using Residue Arithmetic. Communications of the ACM, vol. 14, no. 3 (1971) 180 – 184
25. Jones, N.D., Landweber, L.H., Lien, Y.E.: Complexity of Some Problems in Petri Nets. Theoretical Computer Science, vol. 4 (1977) 277 – 299
26. Chang, Y.-L., Sullivan, R.S.: QS: Quant System, Version 2.1, Prentice-Hall (1996)
27. Watanabe, T.: The Legal Firing Sequence Problem of Petri Nets. IEICE Transactions on Inf. & Sys*t*., vol. E83-D, no. 3 (2000) 397 – 406
28. Huang, J.S., and Murata, T.: A Constructive Method for Finding Legal Transition Sequences in Petri Nets. Journal of Circuits, Systems, and Computers, vol. 8, no. 1 (1998) 189 – 222

Approximability of Bounded Occurrence Max Ones[⋆]

Fredrik Kuivinen[⋆⋆]

Department of Computer and Information Science,
Linköpings Universitet, S-581 83 Linköping, Sweden
freku@ida.liu.se

Abstract. We study the approximability of MAX ONES when the number of variable occurrences is bounded by a constant. For conservative constraint languages (i.e., when the unary relations are included) we give a complete classification when the number of occurrences is three or more and a partial classification when the bound is two. For the non-conservative case we prove that it is either trivial or equivalent to the corresponding conservative problem under polynomial-time many-one reductions.

Keywords: Approximability, Bounded occurrence, Constraint satisfaction problems, Matching, Max Ones.

1 Introduction

Many combinatorial optimisation problems can be formulated as various variants of constraint satisfaction problems (CSPs). MAX ONES is a boolean CSP where we are not only interested in finding a solution but also the measure of the solution. In this paper we study a variant of MAX ONES when the number occurrences of each variable is bounded by a constant.

We denote the set of all n-tuples with elements from $\{0,1\}$ by $\{0,1\}^n$. A subset $R \subseteq \{0,1\}^n$ is a *relation* and n is the *arity* of R. A *constraint language* is a finite set of relations. A constraint language is said to be *conservative* if every unary relation is included in the language. In the boolean case this means that the relations $\{(0)\}$ and $\{(1)\}$ are in the language. The constraint satisfaction problem over the constraint language Γ, denoted CSP(Γ), is defined to be the decision problem with instance (V, C), where V is a set of variables and C is a set of constraints $\{C_1, \ldots, C_q\}$, in which each constraint C_i is a pair (R_i, s_i) with s_i a list of variables of length n_i, called the constraint scope, and R_i an n_i-ary relation over the set $\{0,1\}$, belonging to Γ, called the constraint relation. The question is whether there exists a solution to (V, C) or not. A solution to (V, C) is a function $s : V \to \{0,1\}$ such that, for each constraint $(R_i, (v_1, v_2, \ldots, v_{n_i})) \in C$, the image of the constraint scope is a member of the constraint relation, i.e., $(s(v_1), s(v_2), \ldots, s(v_{n_i})) \in R_i$.

The optimisation problem W-MAX ONES can be defined as follows:

Definition 1 (W-MAX ONES)**.** W-MAX ONES over the constraint language Γ is defined to be the optimisation problem with

[⋆] A full version of this paper is available, see [21].

[⋆⋆] Supported by the *National Graduate School in Computer Science* (CUGS), Sweden.

Instance: Tuple (V, C, w), where (V, C) is an instance of $\textsc{Csp}(\Gamma)$ and $w : V \to \mathbb{N}$ is a function.
Solution: An assignment $f : V \to \{0, 1\}$ to the variables which satisfies the $\textsc{Csp}(\Gamma)$ instance (V, C).
Measure: $\sum_{v \in V} w(v) \cdot f(v)$

The function $w : V \to \mathbb{N}$ is called a *weight function*. In the corresponding unweighted problem, denoted $\textsc{Max Ones}(\Gamma)$, the weight function is restricted to map every variable to 1. The approximability of (W-)MAX ONES has been completely classified by Khanna et al. [19]. Several well-known optimisation problems can be rephrased as (W-)MAX ONES problems, in particular INDEPENDENT SET. We will study W-MAX $\textsc{Ones}(\Gamma)$ with a bounded number of variable occurrences, denoted by W-MAX $\textsc{Ones}(\Gamma)$-k for an integer k. In this problem the instances are restricted to contain at most k occurrences of each variable. The corresponding bounded occurrence variant of $\textsc{Csp}(\Gamma)$ will be denoted by $\textsc{Csp}(\Gamma)$-k.

Schaefer [26] classified the complexity of $\textsc{Csp}(\Gamma)$ for every constraint language Γ. Depending on Γ, Schaefer proved that $\textsc{Csp}(\Gamma)$ is either solvable in polynomial time or is **NP**-complete. The conservative bounded occurrence variant of $\textsc{Csp}(\Gamma)$ has been studied by a number of authors [11,13,14,15]. One result of that research is that the difficult case to classify is when the number of variable occurrences are restricted to two, in all other cases the bounded occurrence problem is no easier than the unrestricted problem. Kratochvíl et al. [20] have studied k-SAT-l, i.e., satisfiability where every clause have length k and there are at most l occurrences of each variable. k-SAT-l is a *non-conservative* constraint satisfaction problem. The complexity classification seems to be significantly harder for such problems compared to the conservative ones. In particular, Kratochvíl et al [20] proves that there is a function f such that k-SAT-l is trivial if $l \leq f(k)$ (every instance has a solution) and **NP**-complete if $l \geq f(k) + 1$. Some bounds of f is given in [20], but the exact behaviour of f is unknown.

$\textsc{Max Ones}(\Gamma)$-$k$ can represent many well-known problems. For $k \geq 3$, we have for example, that INDEPENDENT SET in graphs of maximum degree k is precisely MAX $\textsc{Ones}(\{\{(0,0),(1,0),(0,1)\}\})$-$k$. However, the more interesting case is perhaps $k = 2$ due to its connection to matching problems. (See [25] for definitions and more information about the matching problems mentioned below.) Ordinary weighted maximum matching in graphs is, for example, straightforward to formulate and we get certain generalisations “for free” (because they can be rephrased as ordinary matching problems), such as f-factors and capacitated b-matchings. The general factor problem can also be rephrased as a $\textsc{Max Ones}(\cdot)$-2 problem. A dichotomy theorem for the existence problem of general factors has been proved by Cornuéjols [8]. Some research has also been done on the optimisation problem [7].

In this paper, we start the classification of bounded occurrence MAX ONES. Our first result is a complete classification of W-MAX $\textsc{Ones}(\Gamma)$-k when $k \geq 3$ and $\{(0)\}$ and $\{(1)\}$ are included in Γ. We show that, depending on Γ, this problem is either in **PO**, **APX**-complete or **poly-APX**-complete. Our second result is a partial classification of W-MAX $\textsc{Ones}(\Gamma)$-2. We also give hardness results for the non-conservative case.

The outline of the paper is as follows: in Section 2 we define our notation and present the tools we use. Section 3 and 4 contains our results for three or more occurrences and two occurrences, respectively. Section 5 contains our results for the general case, i.e., when the constraint language is not necessarily conservative. Section 6 contains some concluding remarks. Due to lack of space most of the proofs have been omitted, they are available in the full version of this paper [21].

2 Preliminaries

For an integer n we will use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. The Hamming distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ will be denoted by $d_H(\boldsymbol{x}, \boldsymbol{y})$. For a tuple or vector $\boldsymbol{x}$ the n:th component will be denoted by $\boldsymbol{x}[n]$.

Unless explicitly stated otherwise we assume that the constraint languages we are working with are *conservative*, i.e., every unary relation is a member of the constraint language (in the boolean domain, which we are working with, this means that $\{(0)\}$ and $\{(1)\}$ are in the constraint language).

We define the following relations $NAND^m = \{(x_1, \ldots, x_m) \mid x_1 + \ldots + x_m < m\}$, $EQ^m = \{(x_1, \ldots, x_m) \mid x_1 = x_2 = \ldots = x_m\}$, $IMPL = \{(0,0), (0,1), (1,1)\}$, $c_0 = \{(0)\}$, $c_1 = \{(1)\}$ and the function $h_n(x_1, x_2, \ldots, x_{n+1}) = \bigvee_{i=1}^{n+1}(x_1 \wedge \ldots \wedge x_{i-1} \wedge x_{i+1} \wedge \ldots \wedge x_{n+1})$. For a relation R of arity r, we will sometimes use the notation $R(x_1, \ldots, x_r)$ with the meaning $(x_1, \ldots, x_r) \in R$, i.e., $R(x_1, \ldots, x_r) \iff (x_1, \ldots, x_r) \in R$. If r is the arity of R and $I = \{i_1, \ldots, i_n\} \subseteq [r]$, $i_1 < i_2 < \ldots < i_n$, then we denote the projection of R to I by $R\big|_I$, i.e., $R\big|_I = \{(x_{i_1}, x_{i_2}, \ldots, x_{i_n}) \mid (x_1, x_2, \ldots, x_r) \in R\}$

Representations (sometimes called implementations) have been central in the study of constraint satisfaction problems. We need a notion of representability which is a bit stronger that the usual one, because we have to be careful with how many occurrences we use of each variable.

Definition 2 (k-representable). An n-ary relation R is *k-representable* by a set of relations F if there is a collection of constraints $C_1, \ldots, C_l$ with constraint relations from F over variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ (called *primary variables*) and $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$ (called *auxiliary variables*) such that,

– the primary variables occur at most once in the constraints,
– the auxiliary variables occur at most k times in the constraints, and
– for every tuple $\boldsymbol{z}$, $\boldsymbol{z} \in R$ if and only if there is an assignment to $\boldsymbol{y}$ such that $\boldsymbol{x} = \boldsymbol{z}$ satisfies all of the constraints $C_1, C_2, \ldots, C_l$.

The intuition behind the definition is that if every relation in Γ_1 is k-representable by relations in Γ_2 then W-MAX ONES(Γ_2)-k is no easier than W-MAX ONES(Γ_1)-k. This is formalised in Lemma 6.

2.1 Approximability, Reductions, and Completeness

A *combinatorial optimisation problem* is defined over a set of *instances* (admissible input data) $\mathcal{I}$; each instance $I \in \mathcal{I}$ has a finite set SOL(I) of *feasible solutions* associated with it. The objective is, given an instance I, to find a feasible solution of *optimum* value with respect to some measure function m defined for pairs (x, y) such that $x \in \mathcal{I}$ and $y \in \textsc{sol}(x)$. Every such pair is mapped to a non-negative integer by m. The optimal value is the largest one for *maximisation* problems and the smallest one for *minimisation* problems. A combinatorial optimisation problem is said to be an **NPO** problem if its instances and solutions can be recognised in polynomial time, the solutions are polynomially-bounded in the input size, and the objective function can be computed in polynomial time (see, e.g., [1]).

Definition 3 **(r-approximate).** A solution $s \in \textsc{sol}(I)$ to an instance I of an **NPO** problem Π is *r-approximate* if $\max\left\{\frac{m(I,s)}{\textsc{opt}(I)}, \frac{\textsc{opt}(I)}{m(I,s)}\right\} \leq r$, where $\textsc{opt}(I)$ is the optimal value for a solution to I.

An approximation algorithm for an **NPO** problem Π has *performance ratio* $\mathcal{R}(n)$ if, given any instance I of Π with $|I| = n$, it outputs an $\mathcal{R}(n)$-approximate solution.

Definition 4 **(PO, APX, poly-APX). PO** is the class of **NPO** problems that can be solved (to optimality) in polynomial time. An **NPO** problem Π is in the class **APX** if there is a polynomial-time approximation algorithm for Π whose performance ratio is bounded by a constant. Similarly, Π is in the class **poly-APX** if there is a polynomial-time approximation algorithm for Π whose performance ratio is bounded by a polynomial in the size of the input.

Completeness in **APX** and **poly-APX** is defined using AP-reductions [1]. However, we do not need AP-reductions in this paper, the simpler L- and S-reductions are sufficient for us.

Definition 5 **(L-reduction).** An **NPO** problem Π_1 is said to be *L-reducible* to an **NPO** problem Π_2, written $\Pi_1 \leq_L \Pi_2$, if two polynomial-time computable functions F and G and positive constants β and γ exist such that

- given any instance I of Π_1, algorithm F produces an instance $I' = F(I)$ of Π_2, such that $\textsc{opt}(I') \leq \beta \cdot \textsc{opt}(I)$.
- given $I' = F(I)$, and any solution s' to I', algorithm G produces a solution s to I such that $|m_1(I, s) - \textsc{opt}(I)| \leq \gamma \cdot |m_2(I', s') - \textsc{opt}(I')|$, where m_1 is the measure for Π_1 and m_2 is the measure for Π_2.

It is well-known (see, e.g., Lemma 8.2 in [1]) that, if Π_1 is L-reducible to Π_2 and $\Pi_1 \in$ **APX** then there is an AP-reduction from Π_1 to Π_2.

S-reductions are similar to L-reductions but instead of the condition $\textsc{opt}(I') \leq \beta \cdot \textsc{opt}(I)$ we require that $\textsc{opt}(I') = \textsc{opt}(I)$ and instead of $|m_1(I, s) - \textsc{opt}(I)| \leq \gamma \cdot |m_2(I', s') - \textsc{opt}(I')|$ we require that $m_1(I, s) = m_2(I', s')$. If there is an S-reduction from Π_1 to Π_2 (written as $\Pi_1 \leq_S \Pi_2$) then there is an AP-reduction from Π_1 to Π_2. An **NPO** problem Π is **APX**-*hard* (**poly-APX**-*hard*) if every problem in **APX** (**poly-APX**) is AP-reducible to it. If, in addition, Π is in **APX** (**poly-APX**), then Π is called **APX**-*complete* (**poly-APX**-*complete*).

We will do several reductions from Independent Set (hereafter denoted by MIS) which is **poly-APX**-complete [18]. We will also use the fact that for any $k \geq 3$, MIS restricted to graphs of degree at most k is **APX**-complete [22]. We will denote the latter problem by MIS-k.

The following lemma shows the importance of k-representations in our work.

Lemma 6. *For constraint languages Γ_1 and Γ_2 if every relation in Γ_1 can be k-represented by Γ_2 then* W-MAX ONES(Γ_1)-$k \leq_S$ W-MAX ONES(Γ_2)-k.

Proof. Given an arbitrary instance $I = (V, C, w)$ of W-MAX ONES(Γ_1)-k, we will construct an instance $I' = (V', C', w')$ of W-MAX ONES(Γ_2)-k, in polynomial time. For each $c \in C$, add the k-representation of c to C' and also add all variables which participate in the representation to V' in such a way that the auxiliary variables used in the representation are distinct from all other variables in V'. Let $w'(x) = w(x)$ for all $x \in V$ and $w(x) = 0$ if $x \notin V$ (i.e., all auxiliary variables will have weight zero).

It is not hard to see that: (a) every variable in I' occurs at most k times (b) $\text{OPT}(I') = \text{OPT}(I)$, and (c) given a solution s' to I' we can easily construct a solution s to I (let $s(x) = s'(x)$ for every $x \in V$) such that $m(I, s) = m(I', s')$. Hence, there is an S-reduction from W-MAX ONES(Γ_1)-k to W-MAX ONES(Γ_2)-k. □

2.2 Co-clones and Polymorphisms

Given an integer k, a function $f : \{0,1\}^k \to \{0,1\}$ can be extended to a function over tuples as follows: let $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k}$ be k tuples with n elements each then $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k})$ is defined to be the tuple $(f(\boldsymbol{t_1}[1], \boldsymbol{t_2}[1], \ldots, \boldsymbol{t_k}[1]), \ldots, f(\boldsymbol{t_1}[n], \boldsymbol{t_2}[n], \ldots, \boldsymbol{t_k}[n]))$. Given a n-ary relation R we say that R is *invariant* (or, closed) under f if $\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_k} \in R \Rightarrow f(\boldsymbol{t_1}, \boldsymbol{t_2}, \ldots, \boldsymbol{t_n}) \in R$. Conversely, for a function f and a relation R, f is a *polymorphism* of R if R is invariant under f. For a constraint language Γ we say that Γ is invariant under f if every relation in Γ is invariant under f. We analogously extend the notion of polymorphisms to constraint languages, i.e., a function f is a polymorphism of Γ if Γ is invariant under f. Those concepts has been very useful in the study of the complexity of various constraint satisfaction problems (see, e.g., [16]) and play an important role in this work, too.

The set of polymorphisms for a constraint language Γ will be denoted by $\text{Pol}(\Gamma)$, and for a set of functions C the set of all relations which are invariant under C will be denoted by $\text{Inv}(B)$. The sets $\text{Pol}(\Gamma)$ are *clones* in the sense of universal algebra. For a clone C, $\text{Inv}(C)$ is called a relational clone or a co-clone. Over the boolean domain Emil Post has classified all such co-clones and their inclusion structure in [23].

For a set of relations Γ we define a closure operator $\langle \Gamma \rangle$ as the set of relations that can be expressed with relations from Γ using existential quantification and conjunction (note that we are only allowed to use the relations in Γ, hence equality is not necessarily allowed). Intuitively $\langle \Gamma \cup \{EQ^2\} \rangle$ is the set of relations which can be simulated by Γ in CSP(Γ). An alternative classification of this set is $\langle \Gamma \cup \{EQ^2\} \rangle = \text{Inv}(\text{Pol}(\Gamma))$ [24]. These few paragraphs barely scratch the surface of the rich theory of clones and their relation to the computational complexity of various constraint satisfaction problems, for a more thorough introduction see [3,4,9].

We say that a set of relations B is a *plain basis* for a constraint language Γ if every relation in Γ can be expressed with relations from B using relations from $B \cup \{=\}$ and conjunction. Note that this differs from the definition of the closure operator $\langle \cdot \rangle$ as we do not allow existential quantification. See [10] for more information on plain bases.

We can not only study the co-clones when we try to classify MAX ONES(Γ)-k because the complexity of the problem do not only depend on the co-clone $\langle\Gamma\rangle$. However, the co-clone lattice with the corresponding plain bases and invariant functions will help us in our classification effort. Furthermore, as we mostly study the conservative constraint languages we can concentrate on the co-clones which contain c_0 and c_1. Figure 1 contains the conservative part of Post's lattice and Table 1 contains the plain bases for the relational clones which will be interesting to us (co-clones at and below IV_2 have been omitted as MAX ONES is in **PO** there).

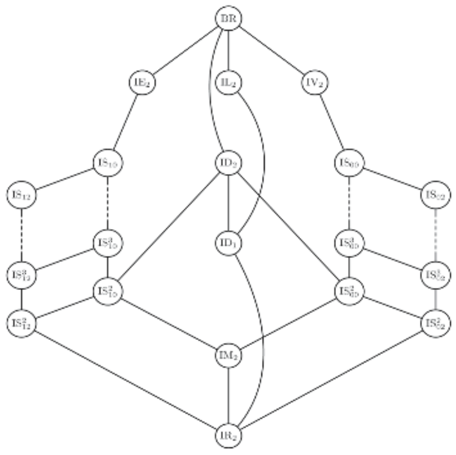


Fig. 1. Lattice of idempotent co-clones

Table 1. Plain bases for some relational clones. The list of plain bases are from [10].[1]

Co-clone	Base for clone	Plain Basis
IE_2	and	$\{N_k \mid k \in \mathbb{N}\} \cup \{(\neg x_1 \vee \ldots \vee \neg x_k \vee y) \mid k \in \mathbb{N}\}$
IS_{10}	$x \wedge (y \vee z)$	$\{c_1, IMPL\} \cup \{N_k \mid k \in \mathbb{N}\}$
IS_{10}^m	$x \wedge (y \vee z), h_n$	$\{c_1, IMPL, N_m\}^{\ddagger}$
IS_{12}	$x \wedge (y \vee \neg z)$	$\{EQ^2, c_1\} \cup \{N_k \mid k \in \mathbb{N}\}$
IS_{12}^m	$x \wedge (y \vee \neg z), h_n$	$\{EQ^2, c_1, N_m\}^{\ddagger}$
IL_2	$x \oplus y \oplus z$	$\{x_1 \oplus \ldots \oplus x_k = c \mid k \in \mathbb{N}, c \in \{0,1\}\}$
ID_2	$xy \vee yz \vee xz$	$\{c_0, c_1, x \vee y, IMPL, NAND^2\}$
ID_1	$xy \vee y(\neg z) \vee y(\neg z)$	$\{c_0, c_1, x \oplus y = 0, x \oplus y = 1\}$
IM_2	and, or	$\{c_0, c_1, IMPL\}$
IR_2	$or, x \wedge (y \oplus z \oplus 1)$	$\{EQ^2, c_0, c_1\}$

3 Three or More Occurrences

In this section we will prove a classification theorem for W-MAX ONES(Γ)-k where $k \geq 3$. The main result of this section is the following theorem.

Theorem 7. *Let Γ be a conservative constraint language and $k \geq 3$,*

1. *If $\Gamma \subseteq IV_2$ then* W-MAX ONES(Γ)-k *is in* **PO**.
2. *Else if $IS_{12}^2 \subseteq \langle\Gamma\rangle \subseteq IS_{12}$ then* (W-)MAX ONES(Γ)-k *is* **APX***-complete if EQ^2 is not k-representable by Γ and* W-MAX ONES(Γ)-k *is* **poly-APX***-complete otherwise.*
3. *Otherwise,* W-MAX ONES(Γ) *and* W-MAX ONES(Γ)-k *are equivalent under S-reductions.*

[1] In [10] the listed plain basis for IS_{12}^m is $\{EQ^2, c_1\} \cup \{N_k | k \leq m\}$ however, if we have N_m then N_{m-1} can be represented without auxiliary variables by $N_{m-1}(x_1, x_2, \ldots, x_{m-1}) \iff N_m(x_1, x_1, x_2, x_3, \ldots, x_{m-1})$, hence the set of relations listed in Table 1 is a plain basis for IS_{12}^m. The same modification has been done to IS_{10}^m.

The first part of Theorem 7 follows from Khanna et al.'s results for MAX ONES [19]. Intuitively the second part follows from the fact that W-MAX ONES($\{NAND^2\}$) is equivalent to MIS, hence if we have access to the equality relation then the problem gets **poly-APX**-complete. On the other hand, if we do not have the equality relation then we essentially get MIS-k, for some k, which is **APX**-complete. The third part follows from Lemmas 8, 9, 10, and 11.

Dalmau and Ford proved the following lemma in [11].

Lemma 8. *If there is a relation R in the constraint language Γ such that $R \notin IE_2$, then either $x \vee y$ or $x \neq y$ can be 3-represented by Γ. By duality, if there is a relation $R \in \Gamma$ such that $R \notin IV_2$, then either $NAND^2$ or $x \neq y$ can be 3-represented.*

We can use the lemma above to get a 3-representation of either EQ^2 or $IMPL$. We will later, in Lemma 11, show that those relations makes the problem as hard as the unbounded occurrence variant.

Lemma 9. *If there is a relation R in the constraint language Γ such that $R \notin IE_2$ and $R \notin IV_2$, then either EQ^2 or $IMPL$ can be 3-represented by Γ.*

Proof. From Lemma 8 we know that either $x \neq y$ or both $x \vee y$ and $NAND^2$ are 3-representable. In the first case $\exists z : \; x \neq z \wedge z \neq y$ is a 3-representation of EQ^2. In the second case $\exists z : \; NAND^2(x, z) \wedge (z \vee y)$ is a 3-representation of $IMPL(x, y)$. □

To get the desired hardness results for the IS_{10} chain we need to prove that we can represent EQ_2 or $IMPL$ in that case too. To this end we have the following lemma.

Lemma 10. *If there is a relation R in the constraint language Γ such that $R \in IE_2$ and $R \notin IS_{12}$, then either EQ^2 or $IMPL$ can be 3-represented by Γ.*

Proof. Let r be the arity of R then, as $R \notin IS_{12}$, there exists a set of minimal cardinality $I \subseteq [r]$, such that $R\big|_I \notin IS_{12}$.

As $g(x, y) = x \wedge y$ is a base of the clone which corresponds to IE_2, $R\big|_I \in IE_2$ implies that g is a polymorphism of $R\big|_I$. Furthermore, as $f(x, y, z) = x \wedge (y \vee \neg z)$ is a base of the clone which corresponds to IS_{12}, $R\big|_I \notin IS_{12}$ implies that f is *not* a polymorphism of $R\big|_I$. Hence, there exists tuples $\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3} \in R\big|_I$ such that $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3}) = \boldsymbol{t} \notin R\big|_I$.

There exists a coordinate $l_1, 1 \leq l_1 \leq r$ such that $(\boldsymbol{t_1}[l_1], \boldsymbol{t_2}[l_1], \boldsymbol{t_3}[l_1]) = (1, 0, 1)$, because otherwise $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3}) = \boldsymbol{t_1}$. Similarly there exists a coordinate $l_2, 1 \leq l_2 \leq r$ such that $(\boldsymbol{t_1}[l_2], \boldsymbol{t_2}[l_2], \boldsymbol{t_3}[l_2])$ is equal to one of $(0, 1, 0)$, $(0, 1, 1)$ or $(1, 0, 0)$. Because otherwise $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3}) = \boldsymbol{t_2}$. From now on, the case $(\boldsymbol{t_1}[l_2], \boldsymbol{t_2}[l_2], \boldsymbol{t_3}[l_2]) = (1, 0, 0)$ will be denoted by (*). Finally, there also exists a coordinate $l_3, 1 \leq l_3 \leq r$ such that $(\boldsymbol{t_1}[l_3], \boldsymbol{t_2}[l_3], \boldsymbol{t_3}[l_3])$ is equal to one of $(0, 0, 1)$, $(0, 1, 1)$, $(1, 0, 0)$, $(1, 0, 1)$ or $(1, 1, 0)$, because otherwise $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3}) = \boldsymbol{t_3}$. The case $(\boldsymbol{t_1}[l_3], \boldsymbol{t_2}[l_3], \boldsymbol{t_3}[l_3]) = (1, 0, 0)$ will be denoted by (**).

As $R\big|_I$ is invariant under g we can place additional restrictions on l_1, l_2 and l_3. In particular, there has to be coordinates l_1, l_2 and l_3 such that we have at least one of the cases (*) or (**), because otherwise $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3}) = g(\boldsymbol{t_1}, \boldsymbol{t_2})$, which is in $R\big|_I$ and we have assumed that $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3}) \notin R\big|_I$. There is no problem in letting $l_2 = l_3$ since we will then get both (*) and (**). This will be assumed from now on. We can also assume, without loss of generality, that $l_1 = 1$ and $l_2 = l_3 = 2$. We can then construct a 3-representation as $R_\phi(x, y) \iff \exists z_3 \ldots z_r : R\big|_I(x, y, z_3, \ldots, z_r) \wedge c_{k_3}(z_3) \wedge c_{k_4}(z_4) \wedge \ldots \wedge c_{k_r}(z_r)$ where $k_i = f(\boldsymbol{t_1}[i], \boldsymbol{t_2}[i], \boldsymbol{t_3}[i])$ for $3 \leq i \leq r$. We will now prove that R_ϕ is equal to one of the relations we are looking for.

If $(0, 1) \in R_\phi$, then we would have $\boldsymbol{t} \in R\big|_I$, which is a contradiction, so $(0, 1) \notin R_\phi$. We will now show that $(0, 0) \in R_\phi$. Assume that $(0, 0) \notin R_\phi$. Then, $R^* = R\big|_{I \setminus \{l_2\}}$ is not in IS_{12} which contradicts the minimality of I. To see this consider the following table of possible tuples in $R\big|_I$,

	$1 = l_1$	$2 = l_2 = l_3$	3	4	$\ldots$
$\boldsymbol{t_1}$	1	1	$\boldsymbol{t_1}[3]$	$\boldsymbol{t_1}[4]$	$\ldots$
$\boldsymbol{t_2}$	0	0	$\boldsymbol{t_2}[3]$	$\boldsymbol{t_2}[4]$	$\ldots$
$\boldsymbol{t_3}$	1	0	$\boldsymbol{t_3}[3]$	$\boldsymbol{t_3}[4]$	$\ldots$
$\boldsymbol{a}$	0	1	$f(\boldsymbol{t_1}[3], \boldsymbol{t_2}[3], \boldsymbol{t_3}[3])$	$f(\boldsymbol{t_1}[4], \boldsymbol{t_2}[4], \boldsymbol{t_3}[4])$	$\ldots$
$\boldsymbol{b}$	0	0	$f(\boldsymbol{t_1}[3], \boldsymbol{t_2}[3], \boldsymbol{t_3}[3])$	$f(\boldsymbol{t_1}[4], \boldsymbol{t_2}[4], \boldsymbol{t_3}[4])$	$\ldots$

We know that $\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3} \in R\big|_I$ and we also know that $\boldsymbol{a} \notin R\big|_I$. Furthermore, if $\boldsymbol{b} \notin R\big|_I$, then $f(\boldsymbol{t_1}, \boldsymbol{t_2}, \boldsymbol{t_3})\big|_{I \setminus \{l_2\}} \notin R^*$ which means that I is not minimal. The conclusion is that we must have $(0, 0) \in R_\phi$. In the same way it is possible to prove that unless $(1, 1) \in R_\phi$, I is not minimal.

To conclude, we have proved that $(0, 0), (1, 1) \in R_\phi$ and $(0, 1) \notin R_\phi$, hence we either have $R_\phi = EQ^2$ or $R_\phi = \{(0, 0), (1, 0), (1, 1)\}$. □

It is now time to use our implementations of EQ^2 or $IMPL$ to prove hardness results. To this end we have the following lemma.

Lemma 11. *If EQ^2 or $IMPL$ is 3-representable by the constraint language Γ then* W-MAX ONES(Γ) $\leq_S$ W-MAX ONES(Γ)-3.

The proof have been omitted from this version. As either EQ^2 or $IMPL$ is available we can construct a cycle of constraints among variables and such a cycle force every variable in the cycle to obtain the same value. Furthermore, each variable occurs only twice in such a cycle so we have one occurrence left for each variable.

4 Two Occurrences

In this section, we study W-MAX ONES(Γ)-2. We are not able to present a complete classification but a partial classification is achieved. We completely classify the co-clones IL_2 and ID_2. For Γ such that $\Gamma \not\subseteq IL_2, ID_2$ we show that if there is a relation which is not a Δ-matroid relation (those are defined below) in Γ then W-MAX ONES(Γ)-2 is **APX**-hard if W-MAX ONES(Γ) is not tractable.

4.1 Definitions and Results

Most of the research done on CSP(Γ)-k (e.g., in [11,13,14]) has used the theory of Δ-matroids. Those objects are a generalisation of matroids and has been widely studied, cf. [5,6]. It turns out that the complexity of W-MAX ONES(Γ)-2 depend to a large degree on if there is a relation which is not a Δ-matroid relation in the constraint language. Δ-matroid relations are defined as follows.

Definition 12 (Δ-matroid relation [11]). Let $R \subseteq \{0,1\}^r$ be a relation. If $\boldsymbol{x}, \boldsymbol{x'} \in \{0,1\}^r$, then $\boldsymbol{x'}$ is a *step from $\boldsymbol{x}$ to $\boldsymbol{y}$* if $d_H(\boldsymbol{x}, \boldsymbol{x'}) = 1$ and $d_H(\boldsymbol{x}, \boldsymbol{x'}) + d_H(\boldsymbol{x'}, \boldsymbol{y}) = d_H(\boldsymbol{x}, \boldsymbol{y})$. R is a *Δ-matroid relation* if it satisfies the following two-step axiom: $\forall \boldsymbol{x}, \boldsymbol{y} \in R$ and $\forall \boldsymbol{x'}$ a step from $\boldsymbol{x}$ to $\boldsymbol{y}$, either $\boldsymbol{x'} \in R$ or $\exists \boldsymbol{x''} \in R$ which is a step from $\boldsymbol{x'}$ to $\boldsymbol{y}$.

As an example of a Δ-matroid relation consider $NAND^3$. It is not hard to see that $NAND^3$ satisfies the two-step axiom for every pair of tuples as there is only one tuple which is absent from the relation. EQ^3 is the simplest example of a relation which is not a Δ-matroid relation. The main theorem of this section is the following partial classification result for W-MAX ONES(Γ)-2. We say that a constraint language Γ *is Δ-matroid* if every relation in Γ is a Δ-matroid relation.

Theorem 13. *Let Γ be a conservative constraint language,*

1. *If $\Gamma \subseteq IV_2$ or $\Gamma \subseteq ID_1$ then* W-MAX ONES(Γ)-2 *is in* **PO**.
2. *Else if $\Gamma \subseteq IL_2$ and,*
   - *Γ is not Δ-matroid then,* W-MAX ONES(Γ)-2 *is* **APX**-*complete.*
   - *otherwise,* W-MAX ONES(Γ)-2 *is in* **PO**.
3. *Else if $\Gamma \subseteq ID_2$ and,*
   - *Γ is not Δ-matroid then,* W-MAX ONES(Γ)-2 *is* **poly-APX**-*complete.*
   - *otherwise,* W-MAX ONES(Γ)-2 *is in* **PO**.
4. *Else if $\Gamma \subseteq IE_2$ and Γ is not Δ-matroid then* W-MAX ONES(Γ)-2 *is* **APX**-*hard.*
5. *Else if Γ is not Δ-matroid then it is* **NP**-*hard to find feasible solutions to* W-MAX ONES(Γ)-2.

Part 1 of the theorem follows from the known results for W-MAX ONES [1]. Part 4 follows from results for CSP(Γ)-2 [13, Theorem 4]. The other parts follows from the results in Sections 4.3 and 4.4 below.

4.2 Tractability Results for W-MAX ONES($\boldsymbol{\Gamma}$)-2

Edmonds and Johnson [12] has shown that the following integer linear programming problem is solvable in polynomial time: maximise $\boldsymbol{wx}$ subject to the constraints $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$, $\boldsymbol{b_1} \leq A\boldsymbol{x} \leq \boldsymbol{b_2}$ and $\boldsymbol{x}$ is an integer vector. Here A is a matrix with integer entries such that the sum of the absolute values of each column is at most 2. $\boldsymbol{b_1}$, $\boldsymbol{b_2}$ and $\boldsymbol{w}$ are arbitrary real vectors of appropriate dimensions. We will denote this problem by ILP-2. With the polynomial solvability of ILP-2 it is possible to prove the tractability of a number of W-MAX ONES(Γ)-2 problems.

4.3 Classification of $\boldsymbol{ID_2}$ and $\boldsymbol{IL_2}$

When Pol(Γ) $=$ Pol(ID_2) or Pol(Γ) $=$ Pol(IL_2) we prove a complete classification result. We start with the hardness results for ID_2, which consists of the following lemma.

Lemma 14. *Let Γ be a constraint language such that* $\mathrm{Pol}(\Gamma) = \mathrm{Pol}(ID_2)$. *If there is a relation $R \in \Gamma$ which is not a Δ-matroid relation, then* W-MAX ONES(Γ)-2 *is* **poly-APX**-*complete.*

The main observations used to prove the lemma is that since $\mathrm{Pol}(\Gamma) = \mathrm{Pol}(ID_2)$ we can 2-represent every two-literal clause. This has been proved by Feder in [13]. Furthermore, if we have access to every two-literal clause and also have a non-Δ-matroid relation then it is possible to make variables participate in three clauses, which was also proved in [13]. The hardness result then follows with a reduction from MIS.

We will use some additional notation in the following proofs. For a tuple $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ and a set of coordinates $A \subseteq [k]$, $\boldsymbol{x} \oplus A$ is defined to be the tuple $(y_1, y_2, \ldots, y_k)$ where $y_i = x_i$ if $i \notin A$ and $y_i = 1 - x_i$ otherwise. We extend this notation to relations: if $R \subseteq \{0,1\}^n$ and $A \subseteq [n]$ then $R \oplus A = \{\boldsymbol{t} \oplus A \mid \boldsymbol{t} \in R\}$.

We will now define a constraint language denoted by $\mathcal{Q}$. We will later prove that W-MAX ONES$(\mathcal{Q})$-2 is in **PO**. $\mathcal{Q}$ is the smallest constraint language such that:

- $\emptyset$, c_0, c_1, EQ^2 and $\{(0,1),(1,0)\}$ are in $\mathcal{Q}$.
- Every relation definable as $\{\boldsymbol{t} \mid d_H(\boldsymbol{0}, \boldsymbol{t}) \leq 1\}$ is in $\mathcal{Q}$.
- If $R, R' \in \mathcal{Q}$ then their cartesian product $\{(\boldsymbol{t}, \boldsymbol{t'}) \mid \boldsymbol{t} \in R, \boldsymbol{t'} \in R'\}$ is also in $\mathcal{Q}$.
- If $R \in \mathcal{Q}$ and n is the arity of R then $R \oplus A \in \mathcal{Q}$ for every $A \subseteq [n]$.
- If $R \in \mathcal{Q}$, n is the arity of R and $f : [n] \rightarrow [n]$ is a permutation on $[n]$ then $\{(t_{f(1)}, t_{f(2)}, \ldots, t_{f(n)}) \mid \boldsymbol{t} \in R\}$ is in $\mathcal{Q}$.

The relation between $\mathcal{Q}$ and the Δ-matroid relations in ID_2 is given by the following lemma.

Lemma 15. *If $R \in ID_2$ is a Δ-matroid relation, then $R \in \mathcal{Q}$.*

As for the tractability part we have the following lemma.

Lemma 16. *Let Γ be a constraint language such that $\Gamma \subseteq ID_2$, if all relations in Γ are Δ-matroid relations then* W-MAX ONES(Γ)-2 *is in* **PO**.

The idea behind the proof is that W-MAX ONES$(\mathcal{Q})$-2 can be seen as an ILP-2 problem and is therefore solvable in polynomial time.

As for IL_2 the result is the same, non Δ-matroids give rise to **APX**-complete problems and absence of such relations makes the problem tractable. Also in this case the tractability follows from a reduction to ILP-2.

4.4 IE_2, IS_{12} and IS_{10}

The structure of the Δ-matroids do not seem to be as simple in IS_{12} and IS_{10} as they are in ID_2 and IL_2. There exists relations in IS_{12} which are Δ-matroid relations but for which we do not know of any polynomial time algorithm. One such relation is $R(x,y,z,w) \iff NAND^3(y,z,w) \wedge NAND^3(x,z,w) \wedge NAND^2(x,y)$. However, we get tractability results for some relations with the algorithm for ILP-2. In particular if the constraint language is a subset of $\{NAND^m \mid m \in \mathbb{N}\} \cup \{IMPL\}$ then W-MAX ONES$(\cdot)$-2 is in **PO**.

We manage to prove hardness results for every non-Δ-matroid relation contained in those co-clones. The main part of our hardness results for the non-Δ-matroid relations is the following lemma.

Lemma 17. *Let* $R(x_1, x_2, x_3) \iff NAND^2(x_1, x_2) \wedge NAND^2(x_2, x_3)$*, then* W-MAX ONES$(\{c_0, c_1, R\})$-2 *is* **APX**-*complete.*

Note that R is not a Δ-matroid relation. With Lemma 17 and a careful enumeration of the types of non-Δ-matroid relations that exists in IE_2, we can deduce the desired result: if there is a non-Δ-matroid relation in the constraint language, then W-MAX ONES$(\cdot)$-2 is **APX**-hard. The proof builds upon the work in [2,13,17].

5 Non-conservative Constraint Languages

In this section we will take a look at the non-conservative case, i.e., we will look at constraint languages which do not necessarily contain c_0 and c_1. A relation R is said to be *1-valid* if it contains the all ones tuple, i.e., R is 1-valid if $(1, 1, \ldots, 1) \in R$. A constraint language is said to be 1-valid if every relation in the language is 1-valid.

Theorem 18. *For any constraint language* Γ *which is not 1-valid, if* W-MAX ONES $(\Gamma \cup \{c_0, c_1\})$-$k$ *is* **NP**-*hard for some integer* k *then so is* W-MAX ONES(Γ)-k.

Note that for constraint languages Γ which are 1-valid W-MAX ONES(Γ) is trivial: the all-ones solution is optimal. The idea in the proof is that we can simulate c_1 constraints by giving the variable a large weight. Furthermore, if there are relations which are not 1-valid then we can represent c_0 constraints when we have access to c_1 constraints. It fairly easy to see why this fails to give us any inapproximability results: due to the large weight used to simulate c_1 any feasible solution is a good approximate solution.

6 Conclusions

We have started the study of the approximability properties of bounded occurrence MAX ONES. We have presented a complete classification for the weighted conservative case when three or more variable occurrences are allowed. Furthermore, a partial classification of the two occurrence case has been presented. In the latter case we have proved that non-Δ-matroid relations give rise to problems which are **APX**-hard if the unbounded occurrence variant is not tractable. We have also given complete classifications for the IL_2 and ID_2 co-clones.

There are still lots of open questions in this area. For example, what happens with the complexity if the weights are removed? Many constraint satisfaction problems such as MAX ONES and MAX CSP do not get any harder when weights are added. Such results are usually proved by scaling and replicating variables and constraints a suitable number of times. However, such techniques do not work in the bounded occurrence setting and we do not know of any substitute which is equally general.

Except for the IS_{12} and IS_{10} chains the open questions in the two occurrence case are certain constraint languages Γ such that Γ only contains Δ-matroid relations and $\mathrm{Pol}(\Gamma) = \mathrm{Pol}(BR)$. It would be very interesting to find out the complexity of W-MAX ONES$(\cdot)$-2 for some of the classes of Δ-matroid relations which have been proved to be tractable for CSP$(\cdot)$-2 in [11,13,14,15]. Instead of trying to classify the entire IS_{12} or IS_{10} chain one could start with IS_{12}^3 or IS_{10}^3. The approximability of the non-conservative case is also mostly open. In light of [20] the computational structure of those problems seems to be quite complex.

References

1. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation*. Springer, 1999.
2. P. Berman and T. Fujito. On the approximation properties of independent set problem in degree 3 graphs. In *WADS '95*, pages 449–460, 1995.
3. E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with boolean blocks, part I: Post's lattice with applications to complexity theory. *SIGACT News*, 34(4):38–52, 2003.
4. E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with boolean blocks, part II: Constraint satisfaction problems. *SIGACT News*, 35(1):22–35, 2004.
5. A. Bouchet. Matchings and Δ-matroids. *Discrete Appl. Math.*, 24(1-3):55–62, 1989.
6. A. Bouchet and W. H. Cunningham. Delta-matroids, jump systems, and bisubmodular polyhedra. *SIAM J. Discret. Math.*, 8(1):17–32, 1995.
7. R. Carr and O. Parekh. A 1/2-integral relaxation for the a-matching problem. *Operations Research Letters*. In Press, Available online 22 September 2005.
8. G. Cornuéjols. General factors of graphs. *J. Comb. Theory Ser. B*, 45(2):185–198, 1988.
9. N. Creignou, S. Khanna, and M. Sudan. *Complexity classifications of boolean constraint satisfaction problems*. Society for Industrial and Applied Mathematics, 2001.
10. N. Creignou, P. Kolaitis, and B. Zanuttini. Preferred representations of boolean relations. Technical Report TR05-119, ECCC, 2005.
11. V. Dalmau and D. Ford. Generalized satisfability with limited occurrences per variable: A study through delta-matroid parity. In *MFCS '03*, pages 358–367, 2003.
12. J. Edmonds and E. L. Johnson. Matching: A well-solved class of integer linear programs. In *Combinatorial Optimization – Eureka, You Shrink!*, pages 27–30, 2001.
13. T. Feder. Fanout limitations on constraint systems. *Theor. Comput. Sci.*, 255(1-2):281–293, 2001.
14. T. Feder and D. Ford. Classification of bipartite boolean constraint satisfaction through delta-matroid intersection. Technical Report TR05-016, ECCC, 2005.
15. G. Istrate. Looking for a version of schaefer's dichotomy theorem when each variable occurs at most twice. Technical Report TR652, University of Rochester Computer Science Department, 1997.
16. P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.
17. V. Kann. Maximum bounded 3-dimensional matching in max snp-complete. *Inf. Process. Lett.*, 37(1):27–35, 1991.
18. S. Khanna, R. Motwani, M. Sudan, and U. V. Vazirani. On syntactic versus computational views of approximability. *SIAM J. Comput.*, 28(1):164–191, 1998.
19. S. Khanna, M. Sudan, L. Trevisan, and D. P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000.
20. J. Kratochvíl, P. Savický, and Z. Tuza. One more occurrence of variables makes satisfiability jump from trivial to np-complete. *SIAM J. Comput.*, 22(1):203–210, 1993.
21. F. Kuivinen. Approximability of bounded occurrence max ones. Technical Report arXiv:cs.CC/0606057, 2006.
22. C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *STOC '88*, pages 229–234. ACM Press, 1988.
23. E. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematical Studies*, 5:1–122, 1941.
24. R. Pöschel and L. Kaluznin. *Funktionen- und Relationenalgebren*. DVW, Berlin, 1979.
25. W. R. Pulleyblank. Matchings and extensions. In *Handbook of combinatorics (vol. 1)*, pages 179–232. MIT Press, 1995.
26. T. J. Schaefer. The complexity of satisfiability problems. In *STOC '78*, pages 216–226. ACM Press, 1978.

Fast Iterative Arrays with Restricted Inter-cell Communication: Constructions and Decidability

Martin Kutrib[1] and Andreas Malcher[2]

[1] Institut für Informatik, Universität Giessen
Arndtstr. 2, D-35392 Giessen, Germany
kutrib@informatik.uni-giessen.de
[2] Institut für Informatik, Johann Wolfgang Goethe Universität
D-60054 Frankfurt am Main, Germany
a.malcher@em.uni-frankfurt.de

Abstract. Iterative arrays (IAs) are one-dimensional arrays of interconnected interacting finite automata with sequential input mode. We investigate IAs which work in real time and whose inter-cell communication is bounded by some constant number of bits not depending on the number of states. It is known [13] that such IAs can recognize rather complicated unary languages with a minimum amount of communication, namely one-bit communication, in real time. Some examples are the languages $\{a^{2^n} \mid n \geq 1\}$, $\{a^{n^2} \mid n \geq 1\}$, and $\{a^p \mid p \text{ is prime}\}$. Here, we consider non-unary languages and it turns out that the non-unary case is quite different. We present several real-time constructions for certain non-unary languages. For example, the languages $\{a^n b^n \mid n \geq 1\}$, $\{a^n (b^n)^m \mid n, m \geq 1\}$, and $\{a^n b a^m b (ba)^{n \cdot m} \mid n, m \geq 1\}$ are recognized in real time by 1-bit IAs. Moreover, it is shown that real-time 1-bit IAs can, in some sense, add and multiply integer numbers. Furthermore, closure properties and decidability questions of communication restricted IAs are investigated. Due to the constructions provided, non-closure results as well as undecidability results can be shown. It turns out that emptiness is still undecidable for 1-bit IAs despite their restricted communication. Thus, also the questions of finiteness, infiniteness, inclusion, and equivalence are undecidable.

1 Introduction

Iterative arrays (IAs) are devices consisting of many identical deterministic finite automata, sometimes called cells, which themselves are homogeneously interconnected with a fixed finite number of neighboring cells. An IA reads the input sequentially via a distinguished communication cell. The state of each cell is changed at discrete time steps by applying its transition function synchronously. Formal language aspects of multidimensional IAs were first studied by Cole in [5] where closure properties of IAs operating in real time are investigated. Furthermore, a technique is developed which allows to show that some languages cannot be accepted by real-time IAs. As a consequence it turned out that the language class accepted by real-time IAs is incomparable with the context-free languages. In [4] it is shown that linear time and two dimensions are sufficient to accept all context-free languages. The ability of real-time IAs to accept rather complicated unary languages was first shown in [6] where a real-time IA for prime numbers is constructed. Other examples are $\{a^{2^n} \mid n \geq 1\}$, $\{a^{n^2} \mid n \geq 1\}$, or $\{a^{n!} \mid n \geq 1\}$. Some recent results concern infinite hierarchies beyond linear time [8] and between real time and linear time [2], hierarchies depending on the amount of nondeterminism [3] and the number of alternating transitions performed by the communication cell [1], and descriptional complexity issues [10].

All these results concern iterative arrays where the states of the neighboring cells are communicated in one time step. That is, the number of bits exchanged is determined by the number of states. A natural and interesting restriction of IAs is to limit the number of bits to some constant being independent of the number of states. Iterative arrays with restricted inter-cell communication have been investigated in [12,13], where algorithmic design techniques for sequence generation are shown. In particular, several important infinite, non-regular sequences such as exponential or polynomial, Fibonacci and prime sequences can be generated in real time. In [14] the computational capacity of one-way cellular automata with restricted inter-cell communication is considered.

In [9] some formal language aspects of IAs with restricted inter-cell communication accepting non-unary languages are investigated. As main results two hierarchies concerning the dimension and the number of bits communicated could be shown. Thus, some questions raised in [12] could be answered. Furthermore, the computational capacity of real-time and linear-time IAs with restricted inter-cell communication has been compared with the power of two-way cellular automata with restricted inter-cell communication. In this paper, these results are extended by some significant results for one-dimensional real-time IAs with restricted inter-cell communication.

2 Definitions and Preliminaries

We denote the rational numbers by $\mathbb{Q}$, the integers by $\mathbb{Z}$, the positive integers and zero $\{0, 1, 2, ...\}$ by $\mathbb{N}$ and the set of positive integers by $\mathbb{N}_+$. The empty word is denoted by λ, the reversal of a word w by w^R, and for the length of w we write $|w|$. The set of words over some alphabet A whose lengths are at most $l \in \mathbb{N}$ is denoted by $A^{\leq l}$. Set inclusion and strict set inclusion are denoted by $\subseteq$ and $\subset$, respectively. By REG we denote the set of regular languages.

A one-dimensional iterative array is a one-dimensional array (i.e. $\mathbb{N}$) of finite automata, sometimes called cells, where each of them is connected to its nearest neighbors. For convenience we identify the cells by their coordinates. Initially they are in the so-called quiescent state. The input is supplied sequentially to the distinguished communication cell at the origin. For this reason, we have two different local transition functions. The state transition of all cells but the communication cell depends on the current state of the cell itself and the current states of its neighbors. The state transition of the communication cell additionally depends on the current input symbol (or if the whole input has been consumed on a special end-of-input symbol). In an iterative array with k-bit restricted inter-cell communication, during every time step each cell may communicate only k bit of information to its neighbors. These bits depend on the current state and are determined by so-called bit-functions. The finite automata work synchronously at discrete time steps.

Definition 1. *A* one-dimensional iterative array with k-bit restricted inter-cell communication (IA$_k$) *is a system* $\langle S, A, F, s_0, k, b_l, b_r, \delta, \delta_0\rangle$, *where*

(1) S *is the finite, nonempty set of* cell states*,*
(2) A *is the finite, nonempty set of* input symbols*,*
(3) $F \subseteq S$ *is the set of* accepting states*,*
(4) $s_0 \in S$ *is the* quiescent state*,*
(6) $k \in \mathbb{N}_+$ *is the* number of bits *which can be* communicated *to neighbor cells,*
(7) $b_l, b_r : S \to \{0,1\}^k$ *are* bit functions *which determine the bits* to communicate *to the left and right neighbors, satisfying* $b_l(s_0) = b_r(s_0) = (0,\ldots,0)$*,*
(8) $\delta : S \times (\{0,1\}^k)^2 \to S$ *is the* local transition function for non-communication cells *satisfying* $\delta(s_0, (0,\ldots,0), (0,\ldots,0)) = s_0$*,*
(9) $\delta_0 : S \times (A \cup \{\#\}) \times \{0,1\}^k \to S$ *is the* local transition function for the communication cell*.*

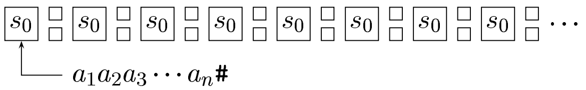


Fig. 1. A one-dimensional one-bit iterative array. The upper box between two cells denotes the bit channel from left to right. The lower box denotes the bit channel from right to left.

Let $\mathcal{M}$ be an IA$_k$. A configuration of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state which is a pair (w_t, c_t), where $w_t \in A^*$ is the remaining input sequence and $c_t : \mathbb{N} \to S$ is a mapping that maps the single cells to their current states. The configuration (w_0, c_0) at time 0 is defined by the input word w_0 and the mapping $c_0(i) = s_0$, $i \in \mathbb{N}$, while subsequent configurations are chosen according to the global transition function Δ. Let (w_t, c_t), $t \geq 0$, be a configuration, then its successor configuration $(w_{t+1}, c_{t+1}) = \Delta\big((w_t, c_t)\big)$ is as follows:

$$c_{t+1}(i) = \delta\big(c_t(i), b_r(c_t(i-1)), b_l(c_t(i+1))\big)$$

for all $i \in \mathbb{N}_+$, and $c_{t+1}(0) = \delta_0(c_t(0), a, b_l(c_t(1)))$ where $a = \#$, $w_{t+1} = \lambda$ if $w_t = \lambda$, and $a = a_1$, $w_{t+1} = a_2 \cdots a_n$ if $w_t = a_1 \cdots a_n$. Thus, the global transition function Δ is induced by δ and δ_0.

A word w is accepted by an IA$_k$ if at some time i during its course of computation on input w the communication cell becomes accepting.

Definition 2. *Let $\mathcal{M} = \langle S, A, F, s_0, k, b_l, b_r, \delta, \delta_0 \rangle$ be an IA_k.*

(1) A word $w \in A^*$ is accepted *by $\mathcal{M}$, if there exists a time step $i \in \mathbb{N}_+$ such that $c_i(0) \in F$.*
(2) $L(\mathcal{M}) = \{w \in A^* \mid w$ is accepted by $\mathcal{M}\}$ is the language accepted *by $\mathcal{M}$.*
(3) Let $t : \mathbb{N}_+ \to \mathbb{N}_+$, $t(n) \geq n+1$, be a mapping. If all $w \in L(\mathcal{M})$ are accepted with at most $t(|w|)$ time steps, then L is said to be of time complexity *t.*

The family of all languages which can be accepted by an IA_k with time complexity t is denoted by $\mathscr{L}_t(\mathrm{IA}_k)$. If t equals the function $n+1$, acceptance is said to be in *real time* and we write $\mathscr{L}_{rt}(\mathrm{IA}_k)$. The *linear-time* languages $\mathscr{L}_{lt}(\mathrm{IA}_k)$ are defined according to $\mathscr{L}_{lt}(\mathrm{IA}_k) = \bigcup_{i \in \mathbb{Q}, i \geq 1} \mathscr{L}_{i \cdot n}(\mathrm{IA}_k)$.

Definition 3. *Let $L \subseteq A^*$ be a language over an alphabet A and $l \in \mathbb{N}_+$ be a constant. Two words $w \in A^{\leq l}$ and $w' \in A^{\leq l}$ are l-left-equivalent with respect to L if for all $y \in A^*$: $wy \in L \iff w'y \in L$. $N_\ell(l, L)$ denotes the number of l-left-equivalence classes with respect to L.*

Lemma 1. *[9] Let $k \in \mathbb{N}_+$ be a constant number. If $L \in \mathscr{L}_t(IA_k)$, then there exists a constant $p \in \mathbb{N}_+$ such that $N_\ell(l, L) \leq p \cdot 2^{k \cdot l}$ for all $l \in \mathbb{N}_+$ and all time complexities $t : \mathbb{N}_+ \to \mathbb{N}_+$.*

3 Constructions

It is known [13] that 1-bit IAs can recognize rather complicated unary languages such as, for example, $\{a^{2^n} \mid n \geq 1\}$, $\{a^{n^2} \mid n \geq 1\}$, or $\{a^p \mid p \text{ is prime}\}$ in real time. The main result in this section is that even 1-bit communication is sufficient to realize binary counters. This enables us to accept rather complicated non-unary languages and to show non-closure and undecidability results in the next sections. We start with a lemma which shows that an additive speed-up in IA_ks is always possible. The benefit of this lemma is that we do not have to care about additive constant factors which may arise in constructions.

Lemma 2. *$\mathscr{L}_{rt+p}(IA_k) = \mathscr{L}_{rt}(IA_k)$ for any constant number $p \in \mathbb{N}_+$.*

Proof. Group p cells into the communication cell and simulate the first p cells there. This enables the communication cell to perform p steps in the $(n+1)$st step at once. □

Example 1. Real-time 1-bit IAs can count by storing the binary encoding of the current value in its cells. The information to be communicated are carry-overs and the position of the most significant bit of the counter. This can be realized by 1-bit IAs: The bit channel from left to right is used to transport carry-overs and the most significant bit of the counter is marked in the bit channel from right to left.

Lemma 3. $\{a^n b^n \mid n \geq 1\} \in \mathscr{L}_{rt}(IA_1)$

Proof. The principal idea is as follows. We implement a binary counter according to Example 1. Then, the counter is increased for every input symbol a. When reading the first b, we send a signal which switches the counter from increasing to decreasing. Then, the counter is decreased for every input symbol b. Finally, when reading the end-of-input symbol, we check whether the counter is zero and accept or reject the input.

To realize the signal, we can observe that the communication cell always alternately sends 1-bits (carry-over) and 0-bits (no carry-over) and that the remaining cells never send two subsequent 1-bits. Thus, two subsequent 1-bits can be considered as signal S. Due to technical reasons two cells are grouped into one cell, so we can speak of pairs. Each coordinate contains the current state of the counter according to Example 1 and is marked $\bullet$ if it is the most significant bit of the counter. Additionally, each cell contains two registers P and R. In P it is marked whether it is counted forwards $\oplus$ or backwards $\ominus$. In R it is marked whether a second 1-bit for transporting the signal S has to be sent to the right.

Now, the binary counter is increased for every input symbol a and P is marked with $\oplus$. If the second coordinate of the communication cell produces a carry-over, the bit channel to the right is set to 1. If any other cell gets a 1-bit from the left, this is interpreted as a carry-over and the binary counter is increased. If the left coordinate of a pair is the most significant bit of the counter, the bit channel to the left is set to 1. So, while reading a's, the counter is increased and the number of cells as well as the rightmost active cell including the most significant bit is extended to the right. If the communication cell reads the first b, counting forwards has to be turned into counting backwards. To this end, the communication cell will send two subsequent 1-bits to the right. To avoid confusion of 1-bits arising from a carry-over with 1-bits arising from the signal S, we introduce four additional time steps in which the communication cell does not process the input. The first two time steps are used to send a possible carry-over from the communication cell to the right. In this case we can observe that there is a 1-bit in the right bit channel at the first time step and a 0-bit in the second one. In the next two time steps the communication cell sends two subsequent 1-bits to the right which will be again sent to the right with the help of the register R and change the register P from $\oplus$ to $\ominus$. Since a 1-bit in the right bit channel is interpreted as carry-over and changes the counter in the appropriate cell, a second 1-bit in the right bit channel, due to the signal S, must cancel the previously made changes. Since two cells are grouped into one cell, this is always possible. Then, we continue to read the input, the binary counter is decreased for every input symbol b and the number of cells as well as the rightmost active cell including the most significant bit is shortened to the left. Finally, when reading the end-of-input symbol it has to be checked whether the counter has been decreased to zero. In this case, the input is accepted and otherwise rejected. It can be observed that the automaton works in real time plus four time steps. Due to Lemma 2 the computation can be sped up to real time. Thus, $\{a^n b^n \mid n \geq 1\} \in \mathscr{L}_{rt}(\mathrm{IA}_1)$. An example schematically accepting a^4b^4 may

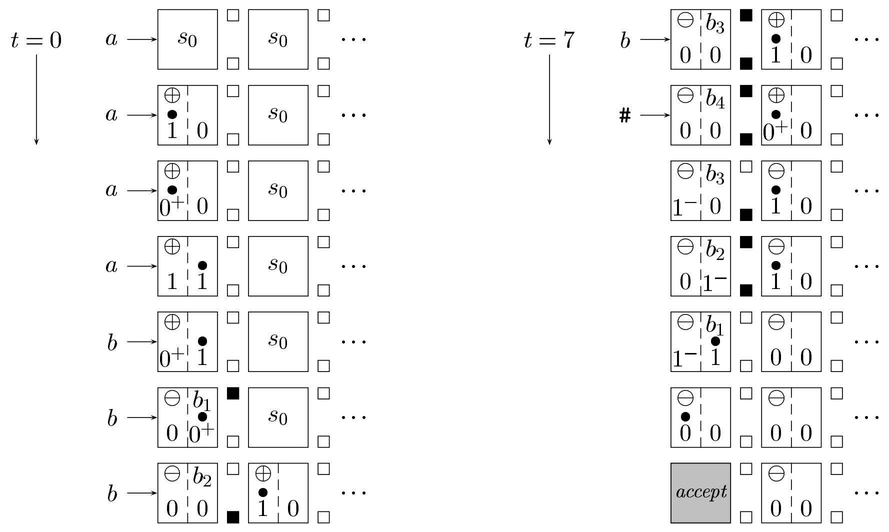


Fig. 2. Schematic computation of a non sped up 1-bit iterative array accepting a^4b^4 in 13 time steps

be found in Fig. 2. It should be added that the main reason for grouping two cells into one was to enable the easy alternation between counting forwards and backwards by using the signal S. It is easily observed that we can realize any number of such alternations by the same construction which will be very useful for the next lemmas. □

Lemma 4. $\{a^n(b^n)^m \mid n, m \geq 1\} \in \mathscr{L}_{rt}(IA_1)$

Proof. 1. Implement two binary counters C_1 and C_2.

2. Increase C_1 and C_2 for every input symbol a.
3. When reading the first b, send a signal which switches C_1 from increasing to decreasing and C_2 to do nothing. The number n is thus binary encoded in C_2.
4. Decrease C_1 for every input symbol b. The current configuration of C_1 can be divided into two parts, namely the active part from the first cell to the cell with the most significant bit and the non-active, remaining cells which are right from the cell with the most significant bit. It can be observed that the latter cells are no longer used to decrement n. Now, the idea is to copy the contents of C_2, i.e. the binary encoding of n, successively to the non-active part of C_1. To this end, we mark some cell i whenever it shifts the most significant bit to its left neighbor. In the next time step, copy the contents of the ith cell of C_2 to the ith cell of C_1.
5. When the first counter is zero, copy the contents of the communication cell of C_2 to the communication cell of C_1. Thus, the binary encoding of n has been copied to C_1. Then, start a new decrement cycle with step 4. in the next time step.
6. When reading the end-of-input symbol, check whether C_1 is zero.

□

The constructions presented in the next two lemmas are necessary to show undecidability results in Section 5. The lemmas show that IA$_1$s can "add" negative and positive integers as well as "multiply" positive integers.

Lemma 5. $\{a^n b^m c^l \mid (n, m, l \geq 1) \wedge (m > n) \wedge (-n + m = l)\} \in \mathscr{L}_{rt}(IA_1)$

Proof. 1. Implement a binary counter.
2. Increase the counter for every input symbol a.
3. When reading the first b, send a signal which switches the counter from increasing to decreasing.
4. Decrease the counter for every input symbol b.
5. If the counter is zero, send a signal which switches the counter from decreasing to increasing.
6. Increase the counter for every further input symbol b.
7. When reading the first c, send another signal which switches the counter from increasing to decreasing.
8. Decrease the counter for every input symbol c.
9. When reading the end-of-input symbol, check whether the counter is zero.

□

Lemma 6. $\{a^n ba^m b(ba)^{n \cdot m} \mid n, m \geq 1\} \in \mathscr{L}_{rt}(IA_1)$

Proof. 1. Implement three binary counters C_1, C_2, C_3 which store n, m, m.
2. While reading $(ba)^*$, alternately look at C_2 (reading a b) and C_1 (reading an a).
3. For every input symbol b, decrease C_2. By the same technique as in Lemma 4, the contents of C_3 are successively copied to the non-active part of C_2.
4. For every input symbol a, check whether C_2 is zero. If so, decrease C_1 by one, copy the contents of the communication cell of C_3 to the communication cell of C_2 and start a new decrement cycle of C_2 in the next time step.
5. When reading the end-of-input symbol, check whether C_1 is zero.

□

An obvious generalization of the construction shows that the following languages can be similarly accepted by IA$_1$s.

– $\{a^n ba^m b(baaaa)^{n^2 \cdot m^3} \mid n, m \geq 1\} \in \mathscr{L}_{rt}(\mathrm{IA}_1)$
– $\{a^n ba^m ba^l b(baaaaaa)^{n^2 \cdot m^3 \cdot l^2} \mid n, m, l \geq 1\} \in \mathscr{L}_{rt}(\mathrm{IA}_1)$

4 Generative Capacity and Closure Properties

In the next theorem we summarize the results concerning the generative capacity of IA$_k$s.

Theorem 1. *For any fixed number $k \in \mathbb{N}_+$ holds*

– $\mathscr{L}_{rt}(IA_k) \subset \mathscr{L}_{rt}(IA_{k+1})$
– $REG \subset \mathscr{L}_{rt}(IA_k) \subset \mathscr{L}_{lt}(IA_k)$.
– $\mathscr{L}_{rt}(IA_k)$ *is incomparable to the context-free languages (CFL)*

Proof. The first two assertions are shown in [9]. It is shown in [5] that there is a language $L \in \text{CFL}$ and $L \notin \mathscr{L}_{rt}(\text{IA})$. Thus, $L \notin \mathscr{L}_{rt}(\text{IA}_k)$. On the other hand, the language $L' = \{a^n ba^m b(ba)^{n \cdot m} \mid n, m \geq 1\} \in \mathscr{L}_{rt}(\text{IA}_k)$ due to Lemma 6, but $L' \notin \text{CFL}$. □

We next investigate the closure properties of the language classes accepted by real-time IA$_k$s and start with positive closure results.

Lemma 7. *$\mathscr{L}_{rt}(IA_k)$ is closed under complementation, intersection with regular languages, union with regular languages, and right concatenation with regular languages.*

Proof. All constructions can be realized in the communication cell and thus are identical to the general case of unrestricted communication [11]. □

It is shown in Lemma 12 that real-time IA$_k$s are not closed under arbitrary inverse homomorphisms. Nevertheless, $\mathscr{L}_{rt}(\text{IA}_k)$ is closed under inverse letter-to-letter homomorphisms. A homomorphism $h : A \to A'$ is called *letter-to-letter* if $h(a) = a'$ for $a \in A$ and $a' \in A'$ where A and A' are two alphabets.

Lemma 8. *$\mathscr{L}_{rt}(IA_k)$ is closed under inverse letter-to-letter homomorphisms.*

Proof. The simulation of the inverse letter-to-letter homomorphism can be realized in the communication cell. □

We next present a language introduced in [9] which will be very useful in the following proofs.

Lemma 9. *[9] For $k \geq 2$ let $A_k = \{a_0, \ldots, a_{2^k-2}\}$. Then a real-time IA_k can be constructed accepting the following language L.*

$$L = \{u_1 \ldots u_m \$ e^{2m+4} \$ e^x \$ e^{2x} \$ v \mid m, x \in \mathbb{N}_+ \textit{ and } x \leq m,$$
$$u_i \in A_k \textit{ with } 1 \leq i \leq m, \textit{ and } v = u_x\}$$

We now turn to non-closure results and first show non-closure under union and intersection. In case of unrestricted communication these operations can be realized using the Cartesian product construction whereas such constructions are not possible in case of restricted communication.

Lemma 10. *Let $k \in \mathbb{N}_+$ with $k \geq 2$. Then $\mathscr{L}_{rt}(IA_k)$ is not closed under intersection and union.*

Proof. Let $A_k = \{a_0, \ldots, a_{2^k-2}\}$ and A'_k be some symbols such that $A_k \cap A'_k = \emptyset$ and $|A_k \cup A'_k| = 2^{k+1}$. Let $M_k = \{M_{1,k}, \ldots, M_{n,k}\}$ be an enumeration of all subsets of $A_k \cup A'_k$ of size $2^k - 1$.

For given $j \in \mathbb{N}_+$ such that $1 \leq j \leq |M_k|$ we consider languages

$$L(j,k) = \{u_1 \ldots u_m \$ e^{2m+4} \$ e^x \$ e^{2x} \$ v \$ w_{j,k} \mid m, x \in \mathbb{N}_+, x \leq m,$$
$$u_i \in M_{j,k} \text{ with } 1 \leq i \leq m, \text{ and } v = u_x\}$$

where $w_{j,k}$ is some fixed word from $M_{j,k}^*$ which enumerates all elements from $M_{j,k}$ and thus identifies the set $M_{j,k}$. By a similar construction as is given in Lemma 9 we obtain that $L(j,k) \in \mathscr{L}_{rt}(\mathrm{IA}_k)$.

We next consider some letter-to-letter homomorphism $h_{j,k} : A_k \cup A'_k \cup \{e, \$\} \longrightarrow M_{j,k} \cup \{e, \$\}$ such that $h_{j,k}(u) = u$ for $u \in M_{j,k} \cup \{e, \$\}$ and define

$$L'(j,k) = h_{j,k}^{-1}(L(j,k)) \cap (A_k \cup A'_k)^* \$ e^* \$ e^* \$ e^* \$ M_{j,k} \$ w_{j,k}$$

and obtain

$$L'(j,k) = \{u_1 \dots u_m \$ e^{2m+4} \$ e^x \$ e^{2x} \$ v \$ w_{j,k} \mid m, x \in \mathbb{N}_+, x \leq m, \\ u_i \in A_k \cup A'_k \text{ with } 1 \leq i \leq m, \text{ and } v = h_{j,k}(u_x) \in M_{j,k}\}$$

Since $\mathscr{L}_{rt}(\mathrm{IA}_k)$ is closed under inverse letter-to-letter homomorphism and intersection with regular sets, we obtain $L'(j,k) \in \mathscr{L}_{rt}(\mathrm{IA}_k)$. For each pair $u_1, u_2 \in A_k \cup A'_k$ with $u_1 \neq u_2$ we next choose some homomorphism $h_{j,k}$ and thus a set $L'(j,k)$ such that $h_{j,k}(u_1) \neq h_{j,k}(u_2)$. We observe that there are at most $\binom{2^{k+1}}{2}$ such sets. Let L_k be the (finite) union of these sets. We next show that $L_k \notin \mathscr{L}_{rt}(\mathrm{IA}_k)$ which implies that $\mathscr{L}_{rt}(\mathrm{IA}_k)$ is not closed under union.

By way of contradiction, we assume that $L_k \in \mathscr{L}_{rt}(\mathrm{IA}_k)$. Due to Lemma 1 there exists a constant $p \in \mathbb{N}_+$ such that $N_\ell(l, L_k) \leq p \cdot 2^{k \cdot l}$, for all $l \in \mathbb{N}_+$.

On the other hand, consider two different words $w = u_1 \cdots u_l$ and $w' = v_1 \cdots v_l$. Since they are different, there is an i such that $u_i \neq v_i$. Furthermore, there is a homomorphism $h_{j,k}$ such that $h_{j,k}(u_i) \neq h_{j,k}(v_i)$. We consider two words $y = w \$ e^{2l+4} \$ e^i \$ e^{2i} \$ h_{j,k}(u_i) \$ w_{j,k}$ and $y' = w' \$ e^{2l+4} \$ e^i \$ e^{2i} \$ h_{j,k}(u_i) \$ w_{j,k}$. Due to the suffix $w_{j,k}$ we know that y and y' are in $L'(j,k)$ if they are in L_k. It is easy to observe that $y \in L_k$ since $h_{j,k}(u_i) = u_i$. On the other hand, we know that $h_{j,k}(v_i) \neq h_{j,k}(u_i)$ which implies that $y' \notin L'(j,k)$. Thus, $y' \notin L_k$. Therefore, we obtain $y \in L_k \iff y' \notin L_k$.

There are $(2^{k+1})^l$ different words of this form. Choosing l such that $2^l > p$, we obtain the following lower bound on the number of induced equivalence classes:

$$N_\ell(l, L_k) \geq (2^{k+1})^l = 2^l 2^{k \cdot l} > p \cdot 2^{k \cdot l}$$

This is a contradiction. Thus, $L_k \notin \mathscr{L}_{rt}(\mathrm{IA}_k)$. Since $\mathscr{L}_{rt}(\mathrm{IA}_k)$ is closed under complementation, $\mathscr{L}_{rt}(\mathrm{IA}_k)$ is not closed under intersection as well. $\square$

To show the non-closure under union for $k = 1$ we cannot apply the above idea directly, because each set $M_{j,k}$ contains only one element. Then each homomorphism $h_{j,k}$ maps $A_1 \cup A'_1$ to only one element. Thus, we cannot find any homomorphism $h_{j,k}$ separating two different elements u_1 and u_2. But some adaptions and finer arguments involving estimations about Fibonacci numbers make the proof also work for the case $k = 1$. The proof is omitted due to space considerations.

Lemma 11. *$\mathscr{L}_{rt}(IA_1)$ is not closed under intersection and union.*

We next show non-closure under arbitrary inverse homomorphisms whereas closure under inverse letter-to-letter homomorphisms is known (Lemma 8). Its proof and the proof of Lemma 13 are omitted due to space considerations. The closure properties of $\mathscr{L}_{rt}(\mathrm{IA}_k)$ are summarized in Table 1.

Table 1. Closure properties of $\mathscr{L}_{rt}(\mathrm{IA}_k)$. $\cap R$ and $\cup R$ denote intersection and union with regular sets. $\cdot R$ and $R\cdot$ denote right and left concatenation with regular sets.

	$\overline{}$	$\cap$	$\cup$	$\cap R$	$\cup R$	$\cdot$	$*$	h	h_λ	h^{-1}	$h^{-1}_{1\text{-}1}$	$\cdot R$	$R\cdot$	^{R}
$\mathscr{L}_{rt}(\mathrm{IA})$	$+$	$+$	$+$	$+$	$+$	$-$	$-$	$-$	$-$	$+$	$+$	$+$	$-$	$-$
$\mathscr{L}_{rt}(\mathrm{IA}_k)$	$+$	$-$	$-$	$+$	$+$	$-$	$-$	$-$	$-$	$-$	$+$	$+$	$-$	$-$
CFL	$-$	$-$	$+$	$+$	$+$	$+$	$+$	$+$	$+$	$+$	$+$	$+$	$+$	$+$

Lemma 12. *$\mathscr{L}_{rt}(IA_k)$ is not closed under inverse homomorphism.*

Lemma 13. *$\mathscr{L}_{rt}(IA_k)$ is not closed under homomorphism, λ-free homomorphism, and reversal. $\mathscr{L}_{rt}(IA_k)$ is not closed under left concatenation with regular languages, concatenation, and Kleene star.*

5 Decidability Questions

For real-time IAs with unrestricted communication it is known that many decidability questions are undecidable [10,11]. One may ask under which additional conditions undecidable questions become decidable. Since the generative capacity of real-time IA_ks is weaker than real-time IAs, it is a natural question whether undecidable questions for real-time IAs are decidable for IA_ks. Furthermore, it is easy to observe that the methods to show undecidability for real-time IAs do not work in case of restricted communication. Nevertheless, in this section we show that all known undecidable questions for real-time IAs remain undecidable for real-time IA_ks. This shows again that even real-time IAs with a minimum amount of communication are very powerful.

We first show that emptiness is undecidable for real-time IA_ks by reduction from the undecidability of Hilbert's tenth problem which is the problem of deciding whether a given polynomial $p(x_1, \dots, x_n)$ with integer coefficients has an integral root. I.e.: Are there integers $\alpha_1, \dots, \alpha_n$ such that $p(\alpha_1, \dots, \alpha_n) = 0$. The same approach has been used in [7] to show that emptiness is undecidable for certain multi-counter machines. As is remarked in [7], it is sufficient to restrict the variables $x_1, \dots, x_n$ to assume non-negative integers only. Such a polynomial has the following form:

$$p(x_1, \dots, x_n) = t_1(x_1, \dots, x_n) + \dots + t_r(x_1, \dots, x_n)$$

where each $t_j(x_1,\ldots,x_n)$ $(1 \le j \le r)$ is a term of the form

$$t_j(x_1,\ldots,x_n) = s_j x_1^{i_{j_1}} \ldots x_n^{i_{j_n}}$$

with $s_j \in \{+1,-1\}$ and $i_{j_1},\ldots,i_{j_n} \ge 0$.

Now, consider the following language given a polynomial $p(x_1,\ldots,x_n)$ with integer coefficients.

$$L(p) = \{a_1^{\alpha_1}\ldots a_n^{\alpha_n}\$(a^{k_1}b)^{\beta_1}\ldots\$(a^{k_r}b)^{\beta_r} \mid \alpha_1,\ldots,\alpha_n \ge 0, k_j = i_{j_1}+\ldots+i_{j_n}, \\ \beta_j = \alpha_1^{i_{j_1}}\ldots\alpha_n^{i_{j_n}}, s_1\beta_1+\ldots+s_r\beta_r = 0\}$$

Lemma 14. *$L(p) \in \mathscr{L}_{rt}(IA_1)$*

Proof. We sketch the construction of a real-time IA$_1$ accepting $L(p)$ which has to calculate $\beta_j = \alpha_1^{i_{j_1}}\ldots\alpha_n^{i_{j_n}}$ for $1 \le j \le r$ and to calculate the sum $s_1\beta_1 + \ldots + s_r\beta_r$. The first calculation, which is a multiplication of positive integers, can be realized with a generalization of Lemma 6. The second calculation is the summation of positive and negative integers which can be realized with a generalization of Lemma 5. The second part of the input is divided into blocks of the form $\$(a^{k_j}b)^{\beta_j}$. It can be observed from Lemma 5 and Lemma 6 that $k_j \cdot \beta_j$ (β_j) time steps are sufficient to perform the first (second) calculation for one block. When reading the end-of-input symbol, it has to be checked whether the current value of the counter representing the second calculation is zero. In this case, the input is accepted and otherwise rejected. $\square$

Theorem 2. *Let $k \ge 1$ be an integer. It is undecidable for an arbitrary real-time IA_k $\mathcal{M}$ whether $L(\mathcal{M}) = \emptyset$.*

Proof. Due to Lemma 14 we can construct a real-time IA$_1$ $\mathcal{M}$ accepting $L(p)$. It is easy to observe that $\mathcal{M}$ accepts the empty set if and only if $p(x_1,\ldots,x_n)$ has no solution in the non-negative integers. Since Hilbert's tenth problem is undecidable, we obtain that the emptiness problem for real-time IA$_1$s is undecidable. $\square$

Corollary 1. *Let $\mathcal{M}$ be a real-time IA_k. The problems finiteness, infiniteness, universality, inclusion, equivalence, regularity, and context-freedom are undecidable for these automata.*

Proof. Consider the language $L(\mathcal{M})\{a\}^*$ for some new alphabet symbol a. Obviously, $L(\mathcal{M})\{a\}^* \in \mathscr{L}_{rt}(\mathrm{IA}_k)$ and $L(\mathcal{M})\{a\}^*$ is finite $\Leftrightarrow L(\mathcal{M}) = \emptyset$. Since emptiness is undecidable, finiteness is undecidable as well. The other claims can be proven similarly. $\square$

The next theorem shows that it is undecidable whether a language accepted by a real-time IA$_{k+1}$ is already accepted by some real-time IA$_k$. The proof is omitted due to space considerations.

Theorem 3. *Let $\mathcal{M}$ be a real-time IA_{k+1}. Then the problem whether $L(\mathcal{M}) \in \mathscr{L}_{rt}(IA_k)$ is undecidable.*

References

1. Buchholz, T., Klein, A., Kutrib, M.: Iterative arrays with a wee bit alternation. In: Ciobanu, G., Păun, G. (eds.): FCT 1999, volume 1684 of LNCS. Springer-Verlag, Berlin (1999) 173–184
2. Buchholz, T., Klein, A., Kutrib, M.: Iterative arrays with small time bounds. In: Nielsen, M., Rovan, B. (eds.): MFCS 2000, volume 1893 of LNCS. Springer-Verlag, Berlin (2000) 243–252
3. Buchholz, T., Klein, A., Kutrib, M.: Iterative arrays with limited nondeterministic communication cell. In: Ito, M., Imaoka, T. (eds.): Words, Languages and Combinatorics III. World Scientific Publishing, Singapore (2003) 73–87
4. Chang, J. H., Ibarra, O. H. , Palis, M. A.: Parallel parsing on a one-way array of finite-state machines. IEEE Transactions Computers **C-36** (1987) 64–75
5. Cole, S. N.: Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE Transactions Computers **C-18** (1969) 349–365
6. Fischer, P. C: Generation of primes by a one-dimensional real-time iterative array. Journal of the ACM **12** (1965) 388–394
7. Ibarra, O. H.: Reversal-bounded multicounter machines and their decision problems. Journal of the ACM **25** (1978) 116–133
8. Iwamoto, C., Hatsuyama, T., Morita, K., Imai, K.: On time-constructible functions in one-dimensional cellular automata. In: Ciobanu, G., Păun, G. (eds.): FCT 1999, volume 1684 of LNCS. Springer-Verlag, Berlin (1999) 317–326
9. Kutrib, M., Malcher, A.: Fast cellular automata with restricted inter-cell communication: computational capacity. Proceedings of IFIP TCS 2006, Santiago de Chile (to appear)
10. Malcher, A.: On the descriptional complexity of iterative arrays. IEICE Transactions on Information Sciences **E87-D** (2004) 721–725
11. Seidel, S. R.: Language recognition and the synchronization of cellular automata. Technical Report 79-02, University of Iowa (1979)
12. Umeo, H., Kamikawa, N.: A design of real-time non-regular sequence generation algorithms and their implementations on cellular automata with 1-bit inter-cell communications. Fundamenta Informaticae **52** (2002) 257–275
13. Umeo, H., Kamikawa, N.: Real-time generation of primes by a 1-bit-communication cellular automaton. Fundamenta Informaticae **58** (2003) 421–435
14. Worsch, T.: Linear Time Language Recognition on Cellular Automata with Restricted Communication. In: Gonnet, G., Panario, D., Viola, A. (eds.): LATIN 2000, volume 1776 of LNCS. Springer-Verlag, Berlin (2000) 417–426

Faster Algorithm for Bisimulation Equivalence of Normed Context-Free Processes

Sławomir Lasota[⋆] and Wojciech Rytter[⋆⋆]

Institute of Informatics, Warsaw University, Warsaw, Poland

Abstract. The fastest known algorithm for checking bisimulation equivalence of normed context-free processes worked in $O(n^{13})$ time. We give an alternative algorithm working in $O(n^8 polylog\ n)$ time, As a side effect we improve the best known upper bound for testing equivalence of simple context-free grammars from $O(n^7 polylog\ n)$ to $O(n^6 polylog\ n)$.

1 Introduction

Equivalence checking, that is determining whether two systems are equal under a given notion of equivalence, is an important verification problem with a long history. In this paper we consider systems described by context-free grammars. It is well known that language equivalence is undecidable in this class [1]. A decidability result was obtained by Korenjak and Hopcroft [12] for a restricted class of deterministic context-free grammars (*simple grammars*). Remarkably, the language containment is undecidable even for simple grammars [6].

In the context of process algebras, a grammar may be considered as a description of a transition graph rather than a language. The adequate concept of equivalence is then *bisimilarity* (bisimulation equivalence), a notion strictly finer than language equivalence. For graphs generated by context-free grammars, called *context-free processes*, bisimilarity is known to be decidable due to the result of [5]. It has also been demonstrated that bisimilarity is the only equivalence in van Glabbeek's spectrum [7] which is decidable for context-free processes. This places bisimilarity in a very favourable position.

Historically the first decision procedure for bisimilarity on infinite-state systems was given by [3] for a class of *normed* context-free processes, those defined by grammars in which, roughly, each nonterminal generates at least one word. Clearly, language equivalence is still undecidable in this class, as normedness assumption does not facilitate testing language equality. As language equivalence and bisimilarity coincide on deterministic graphs (in fact, the whole van Glabbeek's spectrum collapses), the result of [3] was a strict extension of [12].

Later, decidability was extended to all context-free processes [5].

In the case of normed context-free processes, a number of simplifications of the proof of [3] appeared [4,10], relying on particular decomposition properties of

[⋆] Partially supported by the Polish Kbn grant No. 4 T11C 042 25 and by European Community Research Training Network Games.

[⋆⋆] Supported by the Polish Kbn grant No. 4 T11C 044 25.

bisimilarity, and yielding an exponential upper bound for bisimilarity checking. A side effect was an improvement for the equivalence of simple grammars, compared to the complexity of the algorithm of [12] which was $\mathcal{O}(n^v)$, where n is the length of the grammar and v is the length of the shortest word generated, in general exponential in n. Independently, Caucal [4] proposed an algorithm for equivalence of simple grammars working in time $\mathcal{O}(n^3 v)$.

Huynh and Tian [11] did a next step and proved that complexity of bisimilarity is in $\mathrm{NP}^{\mathrm{NP}}$, the second level of the polynomial hierarchy. A first polynomial-time procedure was finally presented by Hirshfeld, Jerrum and Moller [9].

The algorithm in [9] works in time $\mathcal{O}(n^{13})$ and is hence not satisfactory from the practical point of view. This motivated a further research: in [2] an $\mathcal{O}(n^7 \textit{polylog}\, n)$ time algorithm was proposed for the equivalence of simple grammars. In this paper we report a further progress: we give an $\mathcal{O}(n^8 \textit{polylog}\, n)$ time algorithm for bisimilarity on normed context-free processes, thus improving previous $\mathcal{O}(n^{13})$ time of [9]. We believe that our algorithm is conceptually simpler than that of [9]. It is based on the following two insights. First, we avoid an iterative computation of bisimilarity, by a chain of approximants of the greatest fixed point; instead, we are able to reduce the problem of computing the greatest bisimulation to the problem of finding the greatest solution of certain system of boolean equations, and use the linear-time procedure to find this solution. Secondly, we contribute to the algorithmic theory of compressed strings: we develop a fast algorithm for an auxiliary problem on strings called the First Mismatch Problem, working in $\mathcal{O}(n^5 \textit{polylog}\, n)$ time. As a direct corollary, the equivalence of simple grammars can be decided in $\mathcal{O}(n^6 \textit{polylog}\, n)$ time, which beats the complexity of the (fastest known) algorithm of [2].

Context-Free Processes and Bisimilarity. Let Σ be a finite alphabet and $\mathbf{V} = \{X_1, \ldots, X_m\}$ a finite set of variables. By a *process definition* Δ we mean a finite set of rules of the form: $X \xrightarrow{a} \alpha$, with $a \in \Sigma$ and $\alpha \in \mathbf{V}^*$. Such process definitions are usually called in the literature *Basic Process Algebra*, or *Context-Free Processes*. The explanation of the latter is that each rule can be seen as a production $X \longrightarrow a\alpha$ of a context-free grammar in Greibach normal form. Elements of $\mathbf{V}^*$ are called here *processes*; a variable X can be seen as an elementary process.

Δ defines a transition system: its states are processes $\alpha \in \mathbf{V}^*$; and for each $a \in \Sigma$, there is a transition relation containing triples (α, a, β), where $a \in \Sigma$ and $\alpha, \beta \in \mathbf{V}^*$, written $\alpha \xrightarrow{a} \beta$. The transition relations are defined by a prefix rewriting: $X\beta \xrightarrow{a} \alpha\beta$ whenever Δ contains a rule $X \xrightarrow{a} \alpha$, and $\beta \in \mathbf{V}^*$.

Definition 1. *Given a binary relation R over $\mathbf{V}^*$, we say that a pair (α, β) of processes satisfies expansion in R if*

- *whenever $\alpha \xrightarrow{a} \alpha'$, there exists some β' with $\beta \xrightarrow{a} \beta'$ and $(\alpha', \beta') \in R$; and*
- *whenever $\beta \xrightarrow{a} \beta'$, there exists some α' with $\alpha \xrightarrow{a} \alpha'$ and $(\alpha', \beta') \in R$.*

A binary relation S satisfies expansion in R if each pair $(\alpha, \beta) \in S$ does. A relation R is a bisimulation *if it satisfies expansion in itself. We say that α and β are* bisimilar*, denoted by $\alpha \sim \beta$, if (α, β) belongs to some bisimulation.*

Assume that Δ is *normed*, i.e., for each variable $X \in \mathbf{V}^*$ there is a finite sequence $X \xrightarrow{a_1} \alpha_1 \ldots \xrightarrow{a_k} \alpha_k = \varepsilon$ leading from X to the empty process ε. By $|X|$ denote the smallest length of such sequence and call it the *norm* of X (intuitively, $|X|$ is the length of the shortest word generated from X).

We consider the following Normed-BPA-Bisim Problem:

Instance: A normed Δ and $X, Y \in \mathbf{V}$ with $|X| = |Y|$, $X \neq Y$.
Question: Is $X \sim Y$?

A more general problem of checking whether $\alpha \sim \beta$, for any $\alpha, \beta \in \mathbf{V}^*$, can be easily reduced to the above one. We use notation $\widetilde{\mathcal{O}}(f(n))$ for $\mathcal{O}(f(n)\ \mathit{polylog}\, n)$ in the sequel. The size of Δ, denoted by n, is the sum of lengths of all the rules in Δ. Our main result is the following:

Theorem 1. Normed-BPA-Bisim Problem *can be solved in time* $\widetilde{\mathcal{O}}(n^8)$.

2 Terminology and Tools Used in the Main Algorithm

The normedness assumption implies that each variable has at least one rule in Δ. We extend additively the norm to all processes: $|\varepsilon| := 0$, $|X\alpha| := |X| + |\alpha|$. Bisimilarity preserves norm, as a sequence of transitions $X \xrightarrow{a_1} \ldots \xrightarrow{a_k} \varepsilon$ leading to ε must be necessarily matched by a sequence leading to ε as well:

Proposition 1. *If* $\alpha \sim \beta$ *then* $|\alpha| = |\beta|$.

Let Σ, $\mathbf{V}$ and Δ be fixed from now on. We assume that the variables in $\mathbf{V}$ are ordered so that $|X_i| \leq |X_j|$ whenever $i < j$. It is easy to show the following:

Proposition 2. *Norms of all variables can be computed in time* $\widetilde{\mathcal{O}}(n)$.

The following fact is easily derived from the unique decomposition property [9].

Lemma 1. *If* $\alpha\alpha' \sim \beta\beta'$ *and* $|\alpha| \geq |\beta|$ *then for some* γ, $\alpha \sim \beta\gamma$ *and* $\gamma\alpha' \sim \beta'$.

As s direct corollary we get a cancellation property:

Lemma 2. *If* $\gamma\alpha \sim \gamma\beta$ *then* $\alpha \sim \beta$.

We will need a notion of *base*, which is a slight adaptation of [9]. Intuitively, it describes ways of decomposing an elementary process into smaller ones:

Definition 2. *A* **base** *is a set* B *of pairs* $(X_j, X_i\gamma)$, *at most one for each pair* (X_j, X_i), *such that* $i < j$, $\gamma \in \mathbf{V}^*$ *and* $|X_j| = |X_i\gamma|$.

A base is **full** *iff whenever* $X_j \sim X_i\beta$, *for* $j > i$, *then* $(X_j, X_i\gamma) \in B$, *for some* $\gamma \sim \beta$.

Note that whenever $(X_j, X_i\gamma) \in B$ then necessarily $X_i\gamma \in \{X_1, \ldots, X_{j-1}\}^+$. In the sequel we rely on the following lemma proved in [9]:

Lemma 3. *A full base* B_0 *can be constructed, in time* $\mathcal{O}(n^3)$, *such that the length of* γ *is* $\mathcal{O}(n)$, *for each* $(X_j, X_i\gamma) \in B_0$. *Furthermore,* B_0 *contains a pair* $(X_j, X_i\gamma)$ *for each* i, j *such that* $j > i$.

Remark 1. By cancellation, if $X_j \sim X_i\gamma$, then γ is unique up to bisimilarity. Basing on this observation, the construction of B_0 is by inspecting an arbitrarily chosen sequence of $|X_i|$ norm-reducing moves from X_j. Any process obtained at the end of such a sequence is a good candidate for γ.

Remark 2. Note that B_0, being full, contains all pairs (X_j, X_i) with $X_j \sim X_i$.

Fix the full base B_0 from now on. Pairs $(X_j, X_i\gamma) \in B_0$ we call *decomposition pairs*, or *d-pairs* in short. A d-pair $(X_j, X_i\gamma)$ will be denoted by z_{ji}.

In the sequel we will treat the d-pairs as boolean variables. The basic intuition will be that $z_{ji} = \texttt{true}$ just in case when $X_j \sim X_i\gamma$ holds. We will also build the positive boolean formulas on top of d-pairs, by boolean connectives $\wedge$, $\vee$ and symbols `true`, `false` (no negation). The empty conjunction (disjunction) is allowed as a formula and understood as `true` (`false`, respectively).

A *valuation* is a mapping v from B_0 to $\{\texttt{true}, \texttt{false}\}$. We extend valuations to formulas in the obvious way.

Definition 3. *A* boolean equation system *is a set of equations of the form*

$$z_{ji} = \psi_{ji},$$

one for each $z_{ji} \in B_0$, where ψ_{ji} is a positive boolean formula with variables from B_0. A solution *is any valuation v such that $v(z_{ji}) = v(\psi_{ji})$ for each $z_{ji} \in B_0$.*

Valuations are in one-to-one correspondence with subsets of B_0: for $B \subseteq B_0$, a corresponding valuation v_B assigns true to a variable z_{ji} if and only if $z_{ji} \in B$. Each boolean equation system has the greatest solution $\bar{B}$: start with $B = B_0$ and iteratively update B by removing z_{ji} from B if $v_B(\psi_{ji}) = \texttt{false}$, until B eventually stabilizes yielding $\bar{B}$. A relevant observation is a folklore (see e.g. [8]):

Lemma 4. *The greatest solution of a boolean equation system can be computed in time linear wrt. the size of the system.*

The overall idea underlying the algorithm is as follows. Bisimilarity is the greatest bisimulation, i.e., $\sim$ satisfies expansion in itself. Hence, by Knaster-Tarski fixpoint theorem, it is also the greatest fixed point: $\alpha \sim \beta$ if and only if (α, β) satisfies expansion in $\sim$. One crucial insight is that computing this greatest fixed point can be reduced to finding the greatest solution of certain system of boolean equations. Another insight is that the system of equations can be constructed effectively (due to Lemma 5 below). These two insights allowed us to obtain an algorithm working in time $\widetilde{\mathcal{O}}(n^8)$.

Consider any sub-base $B \subseteq B_0$ and two processes $\alpha, \beta \in \mathbf{V}^*$ of equal norm. We write $\alpha =_B \beta$ if α and β can be shown equal by applying any number of substitutions $X_j \longmapsto X_i\gamma$, where $(X_j, X_i\gamma) \in B$. Formally:

Definition 4. *For $B \subseteq B_0$, let $=_B$ be the smallest symmetric relation over processes containing all identical pairs $\alpha =_B \alpha$ and such that if $(X_j, X_i\gamma) \in B$ and $\alpha X_i\gamma\beta =_B \delta$ then $\alpha X_j\beta =_B \delta$.*

Example 1. If $|A| > |B| > |C| > |D| > |E|$ and $B_0 = \{(A, BBD),\ (A, ECEB),\ (C, DE),\ (B, CD), (B, DED),\ \ldots\}$ then we have $AEBBBD =_B BBCCDA$ for $B = \{(A, BBD),\ (B, CD),\ (C, DE)\}$, due to the derivations of the same string:

$$AEBBBD \xrightarrow{A=BBD} BBDEBBBD \xrightarrow{B=CD} BBDECDBBD,$$

$$BBCCDA \xrightarrow{C=DE} BBDECDA \xrightarrow{A=BBD} BBDECDBBD.$$

Note that B is inclusion-minimal, i.e., there is no $B' \subsetneq B$ with $AEBBBD =_{B'} BBCCDA$.

As B_0 contains a pair $(X_j, X_i\gamma)$ for each i and j with $j > i$, it follows that for α, β of equal norm some sub-base B with $\alpha =_B \beta$ always exists; in particular $\alpha =_{B_0} \beta$. The relevant issue will be to find such B possibly small. For future reference, let $B_\sim \subseteq B_0$ be the set of all d-pairs $(X_j, X_i\gamma)$ satisfying $X_j \sim X_i\gamma$.

Definition 5. *We say that B is a* **matching sub-base** *for α, β iff it is an inclusion-minimal subset of B_0 such that (i) $\alpha =_B \beta$ and (ii) $\alpha \sim \beta$ implies $B \subseteq B_\sim$.*

In particular, by inclusion-minimality the matching sub-base for α, α is $\emptyset$. Condition (ii) says that a bisimilar pair can be shown equal by using only bisimilar substitutions. This property will follow by the unique decomposition (cf. Lemma 1) and by our intricate construction of the sub-base in the proof of Lemma 5. The proof of the lemma is postponed to Section 4.

Lemma 5. *For any processes α, β of length $O(n)$ with $|\alpha| = |\beta|$, we can compute in $\widetilde{\mathcal{O}}(n^6)$ time a matching sub-base $B_{\alpha,\beta}$ containing $O(n)$ d-pairs.*

3 The Main Algorithm

Basing on $B_{\alpha,\beta}$, we define a *matching formula for α, β*, denoted by $\phi_{\alpha,\beta}$, as follows: if $|\alpha| = |\beta|$, then $\phi_{\alpha,\beta}$ is a conjunction of all d-pairs z_{ji} belonging to $B_{\alpha,\beta}$, otherwise $\phi_{\alpha,\beta}$ is `false`.

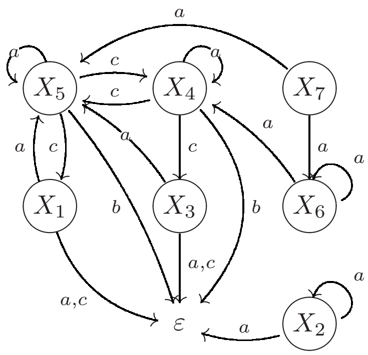
In the algorithm we construct a boolean equation system containing an equation $z_{ji} = \psi_{ji}$ for each z_{ji} and then apply Lemma 4. The intuition is that formula ψ_{ji} expresses the property that the d-pair z_{ji} satisfies expansion in $\sim$. However, instead of directly referring to $\alpha \sim \beta$ in ψ_{ji}, we will prefer to use formulas $\phi_{\alpha,\beta}$ as subformulas in ψ_{ji}, relying on condition (ii) in Definition 5.

Let $z_{ji} = (X_j, X_i\gamma)$. The formula ψ_{ji} is defined as follows:

$$\psi_{ji} \;=\; \bigwedge_a \Big(\bigwedge_\beta \bigvee_\alpha \phi_{\beta,\alpha\gamma} \;\;\wedge\;\; \bigwedge_\alpha \bigvee_\beta \phi_{\beta,\alpha\gamma} \Big), \tag{1}$$

where a ranges over Σ, α ranges over $\{\delta : X_i \xrightarrow{a} \delta \in \Delta\}$ and β ranges over $\{\delta : X_j \xrightarrow{a} \delta \in \Delta\}$.

Example 2. As an illustration, consider Δ containing variables $X_1, \ldots, X_7$ and the rules $X_5 \xrightarrow{c} X_1$, $X_7 \xrightarrow{a} X_5$, etc., as shown in the picture:



Δ is essentially finite-state. All variables have norm 1 except $|X_7| = |X_6| = 2$. Let $B_0 = \{z_{76} = (X_7, X_6), z_{72} = (X_7, X_2X_5), z_{65} = (X_6, X_5X_4),\ z_{64} = (X_6, X_4X_4), z_{62} = (X_6, X_2X_4), z_{54} = (X_5, X_4),\ z_{32} = (X_3, X_2), \ldots\}$. $B_\sim = \{(X_3, X_1), (X_5, X_4), (X_7, X_6), (X_6, X_2X_4), (X_7, X_2X_5)\}$. For instance, the equation for z_{76} is:

$$z_{76} = \big(\phi_{X_5,X_4} \vee \phi_{X_5,X_6}\big) \wedge \big(\phi_{X_6,X_4} \vee \phi_{X_6,X_6}\big) \wedge \big(\phi_{X_6,X_4} \vee \phi_{X_5,X_4}\big) \wedge \big(\phi_{X_6,X_6} \vee \phi_{X_5,X_6}\big).$$

The first conjunct describes possible matchings for $X_7 \xrightarrow{a} X_5$, by $X_6 \xrightarrow{a} X_4$ or $X_6 \xrightarrow{a} X_6$; the second conjunct describes possible matchings for $X_7 \xrightarrow{a} X_6$, etc. Note for instance that $\phi_{X_5,X_4} = z_{54}$. Furthermore $\phi_{X_6,X_6} = \texttt{true}$ as $B_{X_6,X_6} = \emptyset$; and $\phi_{X_5,X_6} = \texttt{false}$ as $|X_5| \neq |X_6|$. After simplification we derive: $z_{76} = z_{54}$. Similarly we derive:

$$z_{54} = z_{54} \wedge \big(z_{51} \vee z_{31}\big) \wedge \big(z_{54} \vee z_{43}\big) \wedge \big(z_{31} \vee z_{43}\big) \wedge \big(z_{51} \vee z_{54}\big).$$

The greatest solution of the equation system is $\{z_{31}, z_{54}, z_{76}, z_{62}, z_{72}\}$.

```
Algorithm Normed-BPA-Bisim($\Delta, X, Y$);    // $|X| = |Y|$
  (1) compute a full base $B_0$;  // c.f. Lemma 3
  (2) for each $(X_j, X_i\gamma) \in B_0$ and $a \in \Sigma$ do
          for each $X_j \xrightarrow{a} \beta$ and $X_i \xrightarrow{a} \alpha$ in $\Delta$ do
              compute the formula $\phi_{\beta,\alpha\gamma}$;
  (3) for each $z_{ji} = (X_j, X_i\gamma) \in B_0$ do
          construct the boolean expression (1);
// this yields a boolean equation system $S$
  (4) compute the greatest solution $\bar{B} \subseteq B_0$ of $S$;
  (5) return $[(X, Y) \in \bar{B}]$.
```

Lemma 6. *The algorithm works in time $\widetilde{\mathcal{O}}(n^8)$.*

Proof. Step (1) requires time $\mathcal{O}(n^3)$, by Lemma 3. The total number of invocations of the procedure computing $\phi_{\beta,\alpha\gamma}$ in step (2) is $\mathcal{O}(n^2)$. Hence, step (2) can be completed in time $\widetilde{\mathcal{O}}(n^8)$, by Lemma 5, and this is dominating the total cost. The size of the boolean equation system built in step (3) is $\mathcal{O}(n^3)$: indeed, the length of each subformula $\phi_{\beta,\alpha\gamma}$ is $\mathcal{O}(n)$, and there is a quadratic number of such formulas in the right-hand sides of equations. The equation system can be constructed in time $\mathcal{O}(n^3)$ and its greatest solution can be computed in the same time, by Lemma 4. □

Lemma 7. *The algorithm is correct.*

Proof. Correctness follows directly from the equality $B_\sim = \bar{B}$, which is shown below in two steps.

We show $B_\sim \subseteq \bar{B}$ first. Denote by ψ_{ji} the right-hand side of equation (1) for $z_{ji} = (X_j, X_i\gamma)$. $\bar{B}$, as the greatest solution of the equation system, is the greatest subset $B \subseteq B_0$ such that $z_{ji} \in B$ (i.e., $v_B(z_{ji}) = \texttt{true}$) implies $v_B(\psi_{ji}) = \texttt{true}$. Hence we will only show that $z_{ji} \in B_\sim$ (i.e, $X_j \sim X_i\gamma$) implies $v_{B_\sim}(\psi_{ji}) = \texttt{true}$.

Indeed. If $X_j \sim X_i\gamma$ then this pair satisfies expansion in $\sim$. Consider any pair $\beta \sim \alpha\gamma$ relevant for the expansion, with $X_j \xrightarrow{a} \beta$ and $X_i \xrightarrow{a} \alpha$ for some a. By point (ii) in Definition 5 we know that all d-pairs appearing in the matching formula $\phi_{\beta,\alpha\gamma}$ are bisimilar. As this applies to any pair $\beta \sim \alpha\gamma$, the right-hand side formula ψ_{ji} is true under valuation $v_{B_\sim}$, as required.

Now we will prove $\bar{B} \subseteq B_\sim$. Consider any solution B of the equation system and any d-pair $z_{ji} = (X_j, X_i\gamma) \in B$. Thus the right-hand side ψ_{ji} evaluates to $\texttt{true}$ under valuation v_B. Consider any matching formula $\phi_{\beta,\alpha\gamma}$ appearing in ψ_{ji}. If it is true under valuation v_B then $B_{\beta,\alpha\gamma} \subseteq B$ and hence $\beta =_B \alpha\gamma$, by point (i) in Definition 5. Hence, by the very construction of ψ_{ji} on top of the matching formulas $\phi_{\beta,\alpha\gamma}$ it follows that $(X_j, X_i\gamma)$ satisfies expansion in $=_B$.

But $=_B$ is clearly contained in $\overset{B}{\equiv}$, the smallest congruence containing B. As the d-pair z was chosen arbitrarily, we have shown that B satisfies expansion in $\overset{B}{\equiv}$, i.e., B is a so called *Caucal base*, or *self-bisimulation* [4]. By a standard argument $\overset{B}{\equiv} \subseteq \sim$, and hence $B \subseteq B_\sim$. As all this was said for an arbitrary solution, in particular $\bar{B} \subseteq B_\sim$ as required. □

4 Computing a Matching Sub-base

This section contains the construction of sub-bases $B_{\alpha,\beta}$ and the proof of Lemma 5. To this aim we need to refine the concept of base a little bit further.

Definition 6. *A production is any pair (X_i, γ), written $X_i \to \gamma$, such that $|X_i| = |\gamma|$ and $\gamma \in \{X_1, \ldots, X_{i-1}\}^+$. A* **decomposition grammar** *(d-grammar) G is any set of productions $X_i \to \gamma$, at most one for each variable X_i.*

Let $\mathbf{V}(G)$ denote the set of all X_i such that $(X_i \to \gamma) \in G$ for some γ. Variables in $\mathbf{V}(G)$ and $\mathbf{V} \setminus \mathbf{V}(G)$ are called *nonterminal* and *terminal symbols*, respectively.

Each nonterminal X_i has precisely one production, hence generates a single nonempty word $G(X_i) \in (\mathbf{V} \setminus \mathbf{V}(G))^+$. We extend this to words $\alpha \in \mathbf{V}^*$ in the obvious way and write $G(\alpha)$ for the single word in $(\mathbf{V} \setminus \mathbf{V}(G))^*$ generated from α. A d-grammar G induces an equivalence $=_G$ over $\mathbf{V}^*$: $\alpha =_G \beta$ iff α and β generate the same word, i.e., $G(\alpha) = G(\beta)$.

Assume $|\alpha| = |\beta|$ hence $|G(\alpha)| = |G(\beta)|$. One of $G(\alpha)$, $G(\beta)$ can not be a proper prefix of the other. Hence, if $\alpha \neq_G \beta$, the words $G(\alpha)$ and $G(\beta)$ must have the left-most mismatching pair of variables. Formally: there is some i, j, γ such that $i \neq j$, γX_i is a prefix of $G(\alpha)$ and γX_j is a prefix of $G(\beta)$. W.l.o.g. assume $i < j$. The pair (X_j, X_i) is called the *first mismatch-pair of α and β wrt. G* and denoted by First-MP(α, β, G). If $|\alpha| = |\beta|$ and $\alpha =_G \beta$, we put First-MP$(\alpha, \beta, G) = \texttt{nil}$. We define First Mismatch Problem:

Input: A d-grammar G and two processes $\alpha, \beta \in \mathbf{V}^*$ of equal norm.
Output: First-MP(α, β, G).

Lemma 8. First Mismatch Problem *can be solved in time $\widetilde{\mathcal{O}}(n^5)$, if the lengths of α, β and all productions $X_i \to \gamma$ in G are in $\mathcal{O}(n)$.*

Section 5 is devoted to the proof of this lemma. In the rest of this section Lemma 8 will be used to prove Lemma 5.

```
Computation of $B_{\alpha,\beta}$;     // $|\alpha| = |\beta|$
    $G := \emptyset$;
    while First-MP$(\alpha, \beta, G) \neq$ nil do
        $(X_j, X_i) :=$ First-MP$(\alpha, \beta, G)$;
        // let $\gamma$ be the unique process such that $(X_j, X_i\gamma) \in B_0$
        $G := G \cup \{X_j \to X_i\gamma\}$;
    $B_{\alpha,\beta} := G$.
```

Note that G is always a d-grammar in the course of the computation, as whenever a production $X_j \to X_i\gamma$ is added to G, X_j has no production yet in G.

For instance, for B_0 and the two processes considered in Example 1 we obtain $B_{AEBBBD,BBCCDA} = \{(A, BBD),\ (C, DE),\ (B, DED)\}$.

Proof (of Lemma 5). The computation of $B_{\alpha,\beta}$ needs $\mathcal{O}(n)$ calls to First Mismatch Problem, hence it completes in $\widetilde{\mathcal{O}}(n^6)$ time.

Now we will show that $B_{\alpha,\beta}$ is a matching sub-base for α, β, in the sense of Definition 5. By the very construction, $B_{\alpha,\beta}$ is an inclusion-minimal d-grammar with $\alpha =_{B_{\alpha,\beta}} \beta$. Furthermore, clearly $B_{\alpha,\beta} \subseteq B_0$. Hence, the equivalence $=_{B_{\alpha,\beta}}$ induced by $B_{\alpha,\beta}$ as a d-grammar is a special case of the relation $=_B$ induced by a sub-base $B \subseteq B_0$, cf. Definition 4. This completes the proof of condition (i) in Definition 5.

For condition (ii), assume $\alpha \sim \beta$. We will show that $B_{\alpha,\beta} \subseteq B_\sim$. It is sufficient to prove that each production $X_j \to X_i\gamma$ added to G in the course of the computation satisfies $X_j \sim X_i\gamma$.

Assume therefore $G \subseteq B_\sim$ (this implies $G(\alpha) \sim G(\beta)$) and $G(\alpha) \neq G(\beta)$. We need to consider the first mismatch-pair of α and β wrt. G, say (X_j, X_i). By Lemma 2 we can ignore the matching prefixes of $G(\alpha)$ and $G(\beta)$, and then by Lemma 1 applied to $\alpha = X_j$ and $\beta = X_i$, we conclude that for some γ' it holds $X_j \sim X_i\gamma'$. Let γ be the unique process for which $(X_j, X_i\gamma) \in B_0$. As B_0 is full, it satisfies $\gamma \sim \gamma'$. As a consequence, $X_j \sim X_i\gamma$ as required. □

5 Algorithm for the First Mismatch Problem

Let G be a given d-grammar. We start with the problem of **equality-testing**: for two nonterminals S_1, S_2 test if they generate the same string. We identify informally the names of nonterminals with their values, so it can be written as $S_1 = S_2$ instead of $S_1 =_G S_2$.

If $A \to A_1A_2\ldots A_r$, then by cut-points, or decomposition points, we mean the positions $|A_1|,\ |A_1|+|A_2|,\ \ldots, |A_1|+|A_2|+|A_3|+\ldots|A_{r-1}|$, see Figure 1.

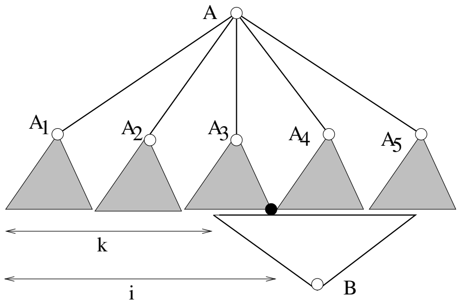


Fig. 1. Assume there is a production $A \to A_1A_2\ldots$. In this case $i = |A_1|+|A_2|+|A_3|$ is the third cut-point of A (black circle) and k is the distance between the possible occurrence of B and the beginning of A. Validity of the overlap item (A, B, i, k) is equivalent to $B = A[k+1\ldots k+|B|]$.

An **overlap item** is a 4-tuple (A, B, i, k) such that i is a decomposition point of A and k is a beginning position of a *potential* occurrence of B in A which overlaps i, see Figure 1. Overlapping means that the occurrence of B is *touching* the point i, i.e., $k \le i \le k+|B| \le |A|$. This overlap item is said to be **valid** iff $B = A[k+1\ldots k+|B|]$.

The equality of two nonterminals $S_1,\ S_2$ is equivalent to the overlap item $\alpha_0 = (S_1, S_2, FirstDecPoint(S_1), 0)$, where $FirstDecPoint(S_1)$ denotes the first decomposition point of S_1. Let us fix in this section S_1, S_2 and α_0.

We say that a set of items $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$ covers an item β iff

$$[\ \alpha_0 \implies (\ \gamma_1\ \&\ \gamma_2\ \&\ \ldots\ \&\ \gamma_p)\ ] \quad \text{and} \quad [\ (\ \gamma_1\ \&\ \gamma_2\ \&\ \ldots\ \&\ \gamma_p\ ) \implies \beta\ ].$$

Observe that if Γ covers α_0 then equality of $S_1 = S_2$ is equivalent to validity of all items in Γ.

Recall that nonterminals are ordered with respect to the increasing norms of their values. If $\mathcal{D}$ is a set of items then $DeleteLexMax(\mathcal{D})$ returns lexicographically maximal element of $\mathcal{D}$ and removes it from $\mathcal{D}$. An item is **atomic** iff the nonterminals occurring in this item generate only terminal symbols. We can test validity of each individual atomic item in constant time.

We describe the basic functions in the equality testing.

The function $SubtleInsert(\beta, \mathcal{D})$ inserts β into $\mathcal{D}$ in $\widetilde{\mathcal{O}}(n)$ time. For every nonterminal B and every cut-point of A we keep only at most three occurrences of B overlapping A on this cut-point. Correctness follows from the fact that the set of occurrence of the same string overlapping a given cut-point is a single arithmetic progression. The function *SubtleInsert* inserts only if it is necessary, and if it inserts β and there are already three occurrences overlapping the same cut-point then one of them is removed.

Next we describe how to implement the function GENERATE. Let α be a non-atomic item. $GENERATE(\alpha)$ is a set of items satisfying the following property: **(1)** it is of size $O(n)$; **(2)** it contains only items lexicographically smaller than α; **(3)** it covers α.

```
function EqTest($S_1, S_2$); // $|S_1| = |S_2|$
  for each production do
      sort the set of its cut-points;
  $\alpha_0 := (S_1, S_2, FirstDecPoint(S_1), 0)$;  $\mathcal{D} := \{\alpha_0\}$;
  while $\mathcal{D}$ contains a non-atomic item do
        $\alpha := DeleteLexMax(\mathcal{D})$;
        for each $\beta \in GENERATE(\alpha)$ $SubtleInsert(\beta, \mathcal{D})$;
  Comment: $\mathcal{D}$ covers $\alpha_0$ and consists only of atomic items;
  return  $[(\forall\ \alpha \in \mathcal{D})\ valid(\alpha)]$
```

Lemma 9. *Let $\alpha = (A, B, i, k)$. We can compute $GENERATE(\alpha)$ satisfying the conditions (1-3) above in $\widetilde{\mathcal{O}}(n)$ time.*

Proof. In the proof we use temporarily other type of items: an *internal item* is a triple (X, Y, t), where t is a potential occurrence of B in A, not necessarily overlapping a cut-point of A. Assume $\alpha = (A, B, i, k)$, and there is a production $B \rightarrow B_1 B_2 \ldots B_r$. We can *locate* each of B_i in A and we have a set of internal items (A, B_1, k), $(A, B_2, k + i_1)$, $\ldots (A, B_r, k + i_1 + \ldots + i_{r-1})$. We can design a subroutine $overlapify(A, B_p, k + i_1 + \ldots + i_{p-1})$, this subroutine finds an overlap item which covers this internal item. It is possible to design such a subroutine $overlapify$ which, applied to all internal items, finds together in $\widetilde{\mathcal{O}}(n)$ time the set of overlap items covering them. Consequently it finds a set of smaller overlap items covering (A, B, i, k). The leftmost and rightmost internal item is *overlapified* in $\widetilde{\mathcal{O}}(n)$ time, all others are processed in logarithmic time, using the sorted order of cut-points and a kind of binary search. This set of overlap items is returned by the function GENERATE. Figure 2 shows how the overlap item $(A, B, *, *)$ is decomposed into the set of internal items $(A, B1, *)$, $(A, B2, *)$, $\dots (A, B6, *)$. The leftmost and rightmost internal items are covered by finding the lowest common ancestors (denoted by X, U in Figure 2) of the endpoints of B1 and B6. This takes $O(n)$ time. All other internal items are covered in logarithmic time, after merging cut-points of B and the set $\mathcal{S}$ (illustrated as small darkened circles in Figure 2) of decomposition points of nonterminals branching from the paths from the root to the nodes X, U. It is crucial that after sorting cut-points for each nonterminal we can preprocess $\mathcal{S}$ in such a way that binary searching (because $\mathcal{S}$ will be sorted) can be done in logarithmic time. The set $\mathcal{S}$ is of size $O(n^2)$ but it consists of $O(n)$ groups of cut-points of nonterminals on the branches from A to X or U. Each group is sorted. Also the beginning and ending positions ($O(n)$ together) of these groups can be first sorted. We omit the details.

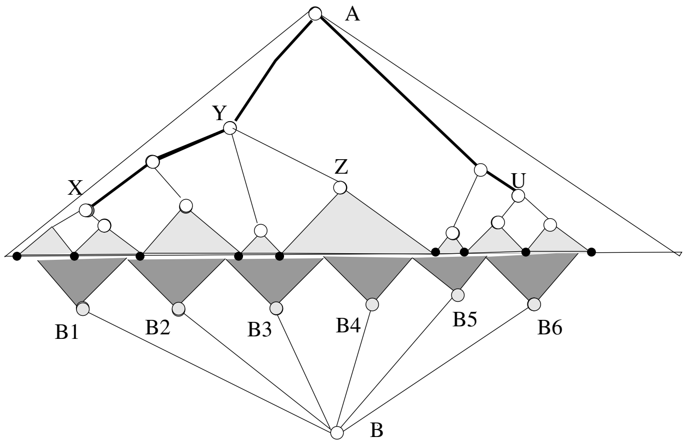


Fig. 2. $(A, B1, *)$ is covered by $(X, B1, *, *)$, $(A, B4, *)$ is covered by $(B, Z, *, *)$ and $(A, B3, *)$ is covered by $(Y, B3, *, *)$, the $*$'s denote corresponding positions in the items. The set $\mathcal{S}$ consists of small darkened circles.

Proof (of Lemma 8). First we analyse the function GENERATE. The total number of overlap items is $\mathcal{O}(n^3)$, we process each item only once with the function GENERATE, it takes $\widetilde{\mathcal{O}}(n)$ time per single item, according to Lemma 9. Altogether the complexity of equality test is $\widetilde{\mathcal{O}}(n^4)$.

Now we can find the first mismatch using the algorithm for equality testing and a kind of binary search. We need at most $O(n)$ instances of equality testing, since the depth of the grammar is $O(n)$. Hence the overall time is $\widetilde{\mathcal{O}}(n^5)$.

6 Equivalence of Simple Grammars

The class of simple grammar is the largest class of context-free grammars for which equivalence problem can be tested in deterministic polynomial time. This is nontrivial since inclusion problem for this class of grammars is undecidable. A simple grammar is a context-free grammar in Greibach normal form, such that whenever $A \rightarrow a\ \alpha$ and $A \rightarrow a\ \beta$ then $\alpha = \beta$. The main component in the $\widetilde{\mathcal{O}}(n^7)$ time algorithm of [2] is the computation of the first mismatch problem in $\widetilde{\mathcal{O}}(n^6)$ time. As we improved this to $\widetilde{\mathcal{O}}(n^5)$ we have immediately the following result.

Theorem 2. *Equivalence of simple grammars can be tested in* $\widetilde{\mathcal{O}}(n^6)$ *time.*

References

1. Y. Bar-Hillel, M. Perles, and S. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift fuer Phonetik, Sprachwissenschaft, und Kommunikationsforschung*, 14:143–177, 1961.
2. C. Bastien, J. Czyżowicz, W. Fraczak, and W. Rytter. Prime normal form and equivalence of simple grammars. In *Proc. CIAA'05*, volume 3845 of *LNCS*, pages 79–90. Springer-Verlag, 2005.
3. J. Beaten, J. Bergstra, and J. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proc. PARLE'87*, volume 259 of *LNCS*, pages 94–113. Springer-Verlag, 1987.
4. D. Caucal. Graphes canoniques des graphes algébraiques. *Informatique Théoretique et Applications (RAIRO)*, 24(4):339–352, 1990.
5. S. Christensen, Y. Hirshfeld, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 12(2):143–148, 1995.
6. E.P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science*, 1:297–316, 1976.
7. R. v. Glabbeek. The linear time - branching time spectrum. In *Proc. CONCUR'90*, pages 278–297, 1990.
8. J. Groote and M. Keinänen. A Sub-quadratic Algorithm for Conjunctive and Disjunctive BESs. CS-Report 04-13, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, June 2004.
9. Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity on normed context-free processes. *Theoretical Computer Science*, 15:143–159, 1996.
10. H. Huettel and C. Stirling. Actions speak louder than words: proving bisimilarity for context-free processes. In *Proc. LICS'91*, pages 376–386. IEEE Computer Society Press, 1991.
11. D. Huynh and L. Tian. Deciding bisimilarity of normed context-free processes is in Σ_2^P. *Theoretical Computer Science*, 123:183–197, 1994.
12. A. Korenjak and J. Hopcroft. Simple deterministic languages. In *Proc. 7th Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46, 1966.
13. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, volume 855 of *LNCS*, pages 460–470. Springer-Verlag, 1994.
14. A. Shinohara, M. Miyazaki, and M. Takeda. An improved pattern-matching for strings in terms of straight-line programs. *Journal of Discrete Algorithms*, 1(1):187–204, 2000.

Quantum Weakly Nondeterministic Communication Complexity

François Le Gall[1,2]

[1] Department of Computer Science, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
[2] ERATO-SORST Quantum Computation and Information Project, JST
Hongo White Building, 5-28-3 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
legall@qci.jst.jp

Abstract. In this paper we study a weak version of quantum nondeterministic communication complexity, corresponding to the most natural generalization of classical nondeterminism, in which a classical proof has to be checked with probability one by a quantum protocol. We prove that, in the framework of communication complexity, even the weak version of quantum nondeterminism is strictly stronger than classical nondeterminism. More precisely, we show the first separation, for a total function, of quantum weakly nondeterministic and classical nondeterministic communication complexity. This separation is quadratic and shows that classical proofs can be checked more efficiently by quantum protocols than by classical ones.

1 Introduction

1.1 Quantum Nondeterminism

Classical nondeterminism, although being an unrealistic model of computation, is a fundamental concept in computational complexity with practical applications, as shown, for example, by the importance of the theory of NP-completeness. There are two different views of classical nondeterminism. A nondeterministic process computing a Boolean function $f(x)$ can be seen as a deterministic process B receiving, besides the input x, a guess, or proof, z and satisfying the following conditions: if $f(x) = 1$ there should exist a proof z such that $B(x, z) = 1$; if $f(x) = 0$ then $B(x, z) = 0$ for all proofs z. Another view of nondeterminism is to consider B receiving no proof, but being probabilistic. Then B should output 1 with positive probability if and only if $f(x) = 1$. It is easy to see that the two models are perfectly equivalent in the classical setting.

These two views of nondeterminism can been extended to obtain two alternative definitions of quantum nondeterminism. The first one, which we call in this paper *quantum strong nondeterminism*, is the quantum version of the probabilistic view of nondeterminism: the quantum process B should output 1 with positive probability if and only if $f(x) = 1$. The second one, which we call *quantum weak nondeterminism*, is the extension of the first view of nondeterminism:

if $f(x) = 1$ there should exist a classical proof z such that $B(x, z) = 1$ with probability 1; if $f(x) = 0$ then $B(x, z) = 0$ with probability 1 for all classical proofs z. In this case, B is thus an exact quantum checking procedure.[1] The point is that, contrary to the classical case, in the quantum setting these two definitions do not seem equivalent and, in the query complexity framework, strong nondeterminism has be shown to be indeed stronger than weak nondeterminism: de Wolf [28] has provided a total function for which the strongly quantum nondeterministic query complexity is $O(1)$, while its quantum weakly nondeterministic query complexity is $\Omega(\sqrt{n})$, where n is the input length.

The main advantages of the strong version of quantum nondeterminism is that the definition is mathematically very convenient and that it leads to many interesting results. For quantum Turing machines, this gives a complexity class known as quantum-NP, which has been shown to be equal to the classical complexity class $co - C_{=}P$ [13,14,29]. For communication protocols, de Wolf [28] has presented an algebraic characterization of quantum strongly nondeterministic communication complexity. Moreover, unbounded ($O(1)$ vs. $\Omega(\log n)$)) and exponential ($O(\log n)$ vs. $\Omega(n)$) gaps are known between quantum strongly nondeterministic and classical nondeterministic communication complexity of some total functions [22,28]. The latter results show the power of quantum strong nondeterminism but, in our opinion, this concept is in a way too powerful to be directly compared with classical nondeterminism. Above all, it lacks the view of nondeterminism as a proof that can be efficiently checked, a view that has been fundamental in complexity theory, for example leading to concepts such as probabilistically checkable proofs (PCP). We refer to [28] for another discussion about these two definitions and a third natural definition where the proof is allowed to be a quantum state, which we will not consider in this paper. We only mention that, although quantum proofs can be extremely useful in some cases (see in particular the works [1,25] studying the power of quantum proofs in certificate complexity and communication complexity, but in the setting where proofs have to be checked only with high probability), as far as quantum weakly nondeterminism is concerned, the proof has to be checked without error and, in this case, the advantage of quantum proofs over classical proofs is not obvious at all.

1.2 Our Contributions

In this paper, we focus on quantum weak nondeterminism and particularly quantum weakly nondeterministic communication complexity, which has, to our knowledge, never been studied before this work. We show a quadratic gap between classical nondeterministic and quantum weakly nondeterministic communication complexity for a total function. We believe that this separation of classical nondeterministic communication complexity and the weakest model

[1] Previous works about quantum procedures receiving proofs focus on randomized (bounded-error) checking [17,21,26]. However, in this paper, even in the quantum setting the proofs have to been checked exactly.

of quantum nondeterministic communication complexity, although being only quadratic, is another indication of the power of quantum computation. Indeed, the proof being classical, such a separation reveals that, if quantum exact checking procedures are allowed, the process of guessing proofs is more powerful than with classical deterministic checking procedures.

Many separations of quantum and classical communication complexity are known in the usual two-players model [2,5,8,16,18,24,28]. In particular, an exponential separation of quantum exact communication complexity and classical nondeterministic communication complexity has been shown for a partial function (i. e. a function where the inputs satisfies a promise) by Buhrman, Cleve and Wigderson [8]. But, except de Wolf's result [28], no gap larger than quadratic between classical and quantum complexity, for any mode of computation, is known for total functions. Moreover, before the present work, the polynomial separations for total functions already found [2,16,8,18] were based on functions related to database search-like problems, which are trivial if classical nondeterminism is allowed, and thus cannot be used to show a gap between quantum weak nondeterminism and classical nondeterminism. The total function we consider in order to show the separation is new, based on the concept of Hadamard codes, and is inspired by a function considered by Buhrman, Fortnow, Newman and Röhrig [9] in the slightly different framework of query complexity and property testing.

We present an efficient quantum weakly nondeterministic protocol computing our function, which generalizes the protocol in [9], based on the local testability property of Hadamard codes and the fact that, with the promise that a string is in the Hadamard code, the string can be decoded efficiently using Bernstein-Vazirani algorithm [6]. The main contribution of our work is the proof of a classical lower bound on the number of bits of communication necessary for any classical nondeterministic protocol that computes our function. This is obtained by showing an upper bound on the number of inputs for which each message can be used, which is basically a problem of extremal combinatorics. This gives a separation $O(\log n)$ vs. $\Omega(\log^2 n)$, where n is the input length, of respectively quantum weakly nondeterministic and classical nondeterministic communication complexity, for our total function.

The paper is structured as follows. We present definitions in Section 2. We then show the quantum upper bound in Section 3 and the classical lower bound in Section 4. Finally, we discuss open problems in Section 5

2 Notations and Definitions

2.1 Notations

In this paper, we will mainly work in vector spaces of the form $\{0,1\}^n$ with the usual addition of vectors $\mathbf{x} = (x_0, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, \ldots, y_{n-1})$ defined as $\mathbf{x} \oplus \mathbf{y} = (x_0 \oplus y_0, \ldots, x_{n-1} \oplus y_{n-1})$, where $x_i \oplus y_i$ denotes the parity of x_i and y_i. The dot product of two vectors $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x} \cdot \mathbf{y} = \bigoplus_{i=0}^{n-1} x_i y_i$. We will in several occasions consider integers in $\{0, \cdots, 2^n - 1\}$ as vectors of $\{0,1\}^n$ through their binary encoding.

We define a function δ over $\mathbb{Z} \times \mathbb{Z}$ as follows.

$$\delta(a,b) = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{if } a \neq b \end{cases}, \text{ for any integers } a \text{ and } b.$$

For $k \geq 1$, we denote by S_k the set $\{1, \cdots, 2^k - 1\} \backslash \{2^j \mid 0 \leq j \leq k-1\}$, i. e. the set of integers in $\{1, \cdots, 2^k - 1\}$ that are not a power of 2. Finally, for any $i \in \{1, \cdots, 2^k - 1\}$, we denote by $[i]$ the largest power of 2 smaller or equal to i. In other words, $[i] = 2^{\lfloor \log_2 i \rfloor}$.

We now recall the definition of Hadamard codes.

Definition 1. For any integer $k \geq 1$, the Hadamard code of length 2^k, denoted $\mathscr{H}_k$, is the set $\{h(\mathbf{w}) \mid \mathbf{w} \in \{0,1\}^k\}$, where $h(\mathbf{w})$ is the binary vector of length 2^k with i-th coordinate $\mathbf{w} \cdot \mathbf{i}$ (for $0 \leq i \leq 2^k - 1$).

Notice that $\mathscr{H}_k$ is a linear code containing 2^k codewords of length 2^k.

2.2 Classical Nondeterministic Communication Complexity

We first recall the definition of classical nondeterministic communication complexity. We refer to the textbook by Hromkovič [20] for further details. Given a set of pairs of strings $X \times Y$, where $X \subseteq \{0,1\}^*$ and $Y \subseteq \{0,1\}^*$, and a function $f : X \times Y \to \{0,1\}$, the communication problem associated to f is the following: Alice has an input $x \in X$, Bob an input $y \in Y$ and their goal is to compute the value $f(x,y)$. We suppose that Alice and Bob have unlimited computation power. Moreover, a proof is given to the protocol: Alice and Bob each receive a string which is private, i. e. each player cannot see the other's part of the proof. We say that a protocol P is a nondeterministic protocol for f if, for each $(x,y) \in X \times Y$, the following holds: if $f(x,y) = 1$ then there is a proof such that the protocol outputs 1; if $f(x,y) = 0$ then, for all proofs, the protocol outputs 0.

The communication complexity of a nondeterministic protocol P that computes correctly f, denoted $N(P,f)$, is the maximum, over all the inputs (x,y) and the proofs, of the number of bits exchanged between Alice and Bob on this input. The nondeterministic communication complexity of the function f, denoted $N(f)$, is the minimum, over all the nondeterministic protocols P that compute f, of $N(P,f)$.

We now recall the notions of rectangle, covering and their relation with classical nondeterministic complexity. A rectangle of $X \times Y$ is a subset $R \subseteq X \times Y$ such that R can be written as $A \times B$ for some $A \subseteq X$ and $B \subseteq Y$. The rectangle R is said to be 1-monochromatic for f if, for all $(x,y) \in R$, $f(x,y) = 1$. A 1-covering of size t for f is a set of t rectangles $R_1, \cdots, R_t$ of $X \times Y$ that are 1-monochromatic for f and such that $R_1 \cup \cdots \cup R_t = \{(x,y) \in X \times Y \mid f(x,y) = 1\}$. Let $C^1(f)$ be the minimum, over all the 1-covering of f, of the size of the covering. Then the following fact holds (we refer to [20] for the proof).

Fact 1. $N(f) = \lceil \log_2 C^1(f) \rceil$.

2.3 Quantum Weakly Nondeterministic Communication Complexity

Let us now consider quantum communication complexity. We refer to Nielsen and Chuang [23] for details about quantum computation and to [7,19,27] for good surveys of quantum communication complexity.

We define quantum weakly nondeterministic protocols as in the classical case, the only modification being that the messages are now allowed to be quantum: Alice and Bob receive inputs x, y and two classical strings corresponding to a classical proof, communicate through a quantum channel and their goal is to compute $f(x,y)$. In this model there is no prior entanglement between the two players. Moreover, we consider the most general setting where both Alice and Bob receive a (private) proof. Actually, the quantum protocol presented in Section 3 can be used even if only one player receives the proof.

Definition 2. We say that such a quantum protocol is a weakly nondeterministic protocol for f if, for each $(x,y) \in X \times Y$, the following holds:

(i) if $f(x,y) = 1$ then there is a classical proof such that the protocol outputs 1 with probability 1;
(ii) if $f(x,y) = 0$ then, for all classical proofs, the protocol outputs 0 with probability 1.

Similarly to the classical case, the quantum weakly nondeterministic communication complexity of f is the minimum, over all the quantum weakly nondeterministic protocols computing f, of the number of quantum bits (qubits) exchanged between Alice and Bob on the worst instance and the worst proof. We are thus considering the worst-case complexity of exact quantum protocols receiving classical proofs.

As explained in the introduction of this paper, a stronger definition of quantum nondeterministic protocols can be given [22,28], corresponding to probabilistic protocols using quantum messages that output 1 with positive probability if and only if $f(x,y) = 1$. The main reasons why we think studying the power of quantum protocols resulting from Definition 2 is meaningful is that, first, this definition corresponds to the original version of classical nondeterminism, based on the notion of proof, and, second, we believe quantum strongly nondeterministic protocols are in a way too powerful to be "fairly" compared with classical nondeterministic protocols. Let us give a simple example that illustrates the latter point. The non-equality function is the function $NEQ_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ such that $NEQ_n(x,y) = 1$ if $x \neq y$ and $NEQ_n(x,y) = 0$ if $x = y$. Massar, Bacon, Cerf and Cleve [22] have shown a quantum strongly nondeterministic protocol for NEQ_n using exactly one qubit of communication. In comparison, it is well known that $N(NEQ_n) = \Theta(\log n)$ (see for example [20]). We explain their simple protocol, which shows the power of quantum strong nondeterminism. Alice sees its input x as an integer in $\{0, \cdots, 2^n - 1\}$, prepares the state

$$\frac{1}{\sqrt{2}}(\cos\left(\frac{x\pi}{2^n}\right)|0\rangle + \sin\left(\frac{x\pi}{2^n}\right)|1\rangle)$$

and sends it to Bob. Bob rotates it by the angle of $-y\pi/2^n$, obtaining the state

$$\frac{1}{\sqrt{2}}(\cos\left(\frac{(x-y)\pi}{2^n}\right)|0\rangle + \sin\left(\frac{(x-y)\pi}{2^n}\right)|1\rangle).$$

Measuring this state gives 1 with positive probability if $x \neq y$. In the case $x = y$, then the probability of measuring 1 is 0. The quantum communication protocol that does the above state manipulations, measures the final state and outputs the outcome of the measurement is thus a quantum strongly nondeterministic communication protocol for NEQ_n using only one qubit of communication, in a way incomparable with classical nondeterminism.

2.4 Our Total Function

We now define the communication problem $HEQ_{k,k'}$ (Hadamard Equality) that is used to show the separation of quantum weakly nondeterministic and classical nondeterministic communication complexity.

Hadamard Equality ($HEQ_{k,k'}$, for $k, k' \geq 1$)

Alice's input: a vector $\mathbf{a} = (a_1, \ldots, a_{2^k-1})$ in $\{0, \cdots, 2^{k'}-1\}^{2^k-1}$

Bob's input: a vector $\mathbf{b} = (b_1, \ldots, b_{2^k-1})$ in $\{0, \cdots, 2^{k'}-1\}^{2^k-1}$

output: $\begin{cases} 0 \text{ if } (0, \delta(a_1, b_1), \ldots, \delta(a_{2^k-1}, b_{2^k-1})) \in \mathscr{H}_k \backslash \{(0, \ldots, 0)\} \\ 1 \text{ else} \end{cases}$

Notice that, for any a and b in $\{0, \cdots, 2^{k'}-1\}$, we have $\delta(a,b) = 0$ if and only if $a = b$. Thus the problem $HEQ_{k,k'}$ can be seen as a two-leveled string equality problem: the hard case is for Alice and Bob to check whether $(0, \delta(a_1, b_1), \ldots, \delta(a_{2^k-1}, b_{2^k-1})) = (0, \ldots, 0)$ and, to do this, they have, intuitively, to check whether $\delta(a_i, b_i) = 0$ for at least k different values of the index i. The point is that the nondeterministic communication complexity of testing the equality of two integers of k' bits is $\Theta(k')$. Thus, intuitively, the classical nondeterministic communication complexity of $HEQ_{k,k'}$ is $\Omega(kk')$. We will, in Section 4, prove that this intuition is correct.

To our knowledge, the function $HEQ_{k,k'}$ has never been considered before, but the case $k' = 1$ is similar to a property testing problem considered by Buhrman, Fortnow, Newman and Röhrig [9] in the framework of query complexity. The original (promise) problem in [9] is, for a fixed subset A_k of $\mathscr{H}_k$, to decide whether a string x is in A_k or the Hamming distance between x and any string of A_k is sufficiently large, by querying as few bits of x as possible. By setting $A_k = \mathscr{H}_k \backslash \{(0, \ldots, 0)\}$, and replacing "sufficiently large" by "positive", we obtain a definition similar to $HEQ_{k,1}$. However, as far as communication complexity is concerned, the results in [9] do not imply any separation of classical nondeterminism and quantum weak nondeterminism.

3 Quantum Upper Bound

In this section, we present an efficient quantum weakly nondeterministic protocol for $HEQ_{k,k'}$. We first prove the following lemma, which restates, in our notations, a well-known property of the Hadamard code.

Lemma 1. *Let* $\mathbf{x} = (x_0, x_1, \ldots, x_{2^k-1})$ *be a vector in* $\{0,1\}^{2^k}$ *such that* $x_0 = 0$. *Then* $\mathbf{x} \in \mathscr{H}_k$ *if and only if* $x_i = x_{[i]} \oplus x_{i-[i]}$ *for all the indexes* i *in* S_k.

Proof. Take a vector $\mathbf{x} \in \mathscr{H}_k$ and an integer i in S_k. ¿From the definition of the Hadamard code, there exists a vector $\mathbf{w} \in \{0,1\}^k$ such that $x_i = \mathbf{w}\cdot\mathbf{i}$, $x_{[i]} = \mathbf{w}\cdot\mathbf{i}'$ and $x_{i-[i]} = \mathbf{w}\cdot\mathbf{i}''$, with $i' = [i]$ and $i'' = i-[i]$. Then $x_{[i]} \oplus x_{i-[i]} = \mathbf{w}\cdot(\mathbf{i}'\oplus\mathbf{i}'') = \mathbf{w}\cdot\mathbf{i}$ from the definition of $[i]$.

Now we prove that there exist at most 2^k vectors in $\{0,1\}^{2^k}$ satisfying this property. Since $|\mathscr{H}_k| = 2^k$, this will conclude the proof. Take two vectors $\mathbf{x}$ and $\mathbf{x}'$ such that $x_0 = x'_0 = 0$ and $x_{2^l} = x'_{2^l}$ for all $l \in \{0,\ldots,k-1\}$. If $\mathbf{x}$ and $\mathbf{x}'$ both satisfy the property, then the other bits are uniquely determined and thus, necessarily, $\mathbf{x} = \mathbf{x}'$. This implies that we can construct at most 2^k different vectors satisfying this property. □

We then present the main result of this section.

Theorem 1. *For any positive integers* k *and* k', *there exists a quantum weakly nondeterministic protocol using less than* $3(k+k')$ *qubits of communication that computes the function* $HEQ_{k,k'}$.

Proof. We describe our quantum protocol, which is actually a generalization of (a modified version of) the quantum query protocol in [9]. Suppose that the inputs are $\mathbf{a} = (a_1,\ldots,a_{2^k-1})$, $\mathbf{b} = (b_1,\ldots,b_{2^k-1})$ and that $(\mathbf{a},\mathbf{b})$ is a 1-instance of $HEQ_{k,k'}$. This means that one of the two following cases holds:

(i) $(0, \delta(a_1,b_1),\ldots,\delta(a_{2^k-1},b_{2^k-1})) \notin \mathscr{H}_k$; or
(ii) $(0, \delta(a_1,b_1),\ldots,\delta(a_{2^k-1},b_{2^k-1})) = (0,\ldots,0)$.

Alice first guesses which case holds. If (i) really holds then, from Lemma 1, there exists an integer $j \in S_k$ such that $\delta(a_j,b_j) \neq \delta(a_{[j]},b_{[j]}) \oplus \delta(a_{j-[j]},b_{j-[j]})$. Alice guesses this index j, sends the value of her guess j and the three integers a_j, $a_{[j]}$ and $a_{j-[j]}$ (using a classical message). Bob then checks whether $\delta(a_j,b_j) \neq \delta(a_{[j]},b_{[j]}) \oplus \delta(a_{j-[j]},b_{i-[j]})$, outputs 1 if it holds, and 0 else.

Now suppose that Alice guessed that (ii) holds. Alice then creates and sends Bob the state

$$\frac{1}{\sqrt{2^k}}\sum_{m=0}^{2^k-1}|m\rangle|a_m\rangle,$$

where the first register consists in k qubits and the second register k' qubits. Here, we use the convention $a_0 = 0$. Bob applies on the state he received the unitary transform

$$|m\rangle|r\rangle \mapsto (-1)^{\delta(r,b_m)}|m\rangle|r\rangle,$$

for all $m \in \{0, \cdots, 2^k - 1\}$ and $r \in \{0, \cdots, 2^{k'} - 1\}$, with the convention $b_0 = 0$. He then sends back the resulting state to Alice. Alice now performs the unitary transform

$$|m\rangle|r\rangle \mapsto |m\rangle|r \oplus a_m\rangle,$$

for any $m \in \{0, \cdots, 2^k - 1\}$ and $r \in \{0, \cdots, 2^{k'} - 1\}$. Here $r \oplus a_m$ denote the bitwise parity of the binary encodings of r and a_m. The resulting state is

$$\frac{1}{\sqrt{2^k}} \sum_{m=0}^{2^k-1} (-1)^{\delta(a_m, b_m)} |m\rangle|0\rangle.$$

From now, it is simply Bernstein-Vazirani algorithm [6] (or Deutsch-Jozsa algorithm [12]). Alice applies a Hadamard transform on each of the k qubits of the first register and measures the first register of the resulting state in the computational basis, outputs 1 if the result is 0 and outputs 0 else. If (ii) really holds, the state before the measurement being $|0\rangle|0\rangle$, her measurement result is necessarily 0. She then outputs 1 without error. For any 1-instance of $HEQ_{k,k'}$, there is thus a guess that can be verified with probability 1 by this protocol.

Now consider the behavior of this protocol on a 0-instance, i. e. an instance such that $(0, \delta(a_1, b_1), \ldots, \delta(a_{2^k-1}, b_{2^k-1})) \in \mathscr{H}_k \backslash \{(0, \ldots, 0)\}$. If Alice guesses that the case (i) holds, then, from Lemma 1, the checking procedure always outputs 0. If Alice guesses that the case (ii) holds, then at the end of the checking procedure, before doing the measurement, the state will be $|c\rangle|0\rangle$ for some $c \in \{1, \cdots, 2^k - 1\}$. Measuring this state will give c which is different from 0. Thus the checking procedure outputs 0 with probability 1, whatever Alice's guesses are. We conclude that the above protocol is correct on 0-instances as well. □

4 Classical Lower Bound

First, notice that there exists a nondeterministic classical protocol for $HEQ_{k,k'}$ using $O(kk')$ communication bits. The protocol is similar to the quantum protocol of Theorem 1, but, when Alice guesses that $(0, \delta(a_1, b_1), \ldots, \delta(a_{2^k-1}, b_{2^k-1})) = (0, \ldots, 0)$, she sends the k integers a_{2^s}, for all $s \in \{0, \cdots, k-1\}$, instead of sending the state $\frac{1}{\sqrt{2^k}} \sum_{m=0}^{2^k-1} |m\rangle|a_m\rangle$. Bob then outputs 1 if and only if $\delta(a_{2^s}, b_{2^s}) = 0$ for all these integers s. The objective of this section is to show that this protocol is basically optimal.

The proof of the lower bound is based on the following result.

Theorem 2. *Let k and k' be two positive integers such that $k \geq 3$ and $k' \geq k$. Consider any subset $A \subseteq \{0, \cdots, 2^{k'} - 1\}^{2^k-1}$ such that, for any two elements $\mathbf{a} = (a_1, \ldots, a_{2^k-1})$ and $\mathbf{b} = (b_1, \ldots, b_{2^k-1})$ of A, the following condition holds.*

$$\begin{cases} (0, \delta(a_1, b_1), \ldots, \delta(a_{2^k-1}, b_{2^k-1})) = (0, \ldots, 0) & \textit{if } \mathbf{a} = \mathbf{b} \\ (0, \delta(a_1, b_1), \ldots, \delta(a_{2^k-1}, b_{2^k-1})) \notin \mathscr{H}_k & \textit{if } \mathbf{a} \neq \mathbf{b} \end{cases} \tag{1}$$

Then A necessarily satisfies

$$|A| \leq 2^{k'2^k - k(k'-k-1)}.$$

Proof. Our proof is inspired by a new proof by Babai, Snevily and Wilson [4] of a result by Frankl [15], itself generalizing a result by Delsarte [10,11], which gives an upper bound on the size of any code in function of the cardinality of the set of Hamming distances that occurs between two distinct codewords (but these results are basically different from what we need to prove our upper bound).

Denote $\mathscr{A} = \{0, \cdots, 2^{k'}-1\}^{2^k-1}$, and consider any subset $A \subseteq \mathscr{A}$ such that any two elements $\mathbf{a}$ and $\mathbf{b}$ of A satisfy Equation (1). For each $a \in \{0, \cdots, 2^{k'}-1\}$, consider the polynomial ε_a over the field of rational numbers defined as follows.

$$\varepsilon_a(X) = 1 - \frac{X}{a}\,\frac{X-1}{a-1}\cdots\frac{X-(a-1)}{1}\,\frac{X-(a+1)}{-1}\cdots\frac{X-(2^{k'}-1)}{a-(2^{k'}-1)}$$

Notice that $\varepsilon_a(b) = \delta(a,b)$ for any a and b in $\{0, \cdots, 2^{k'}-1\}$. Now, given a vector $\mathbf{a} = (a_1, \ldots, a_{2^k-1})$ in $\mathscr{A}$, we define the multivariate polynomial

$$f_{\mathbf{a}}(\mathbf{X}) = f_{\mathbf{a}}(X_1, \ldots, X_{2^k-1}) = \prod_{i \in S_k} \left(1 - \varepsilon_{a_i}(X_i) - \varepsilon_{a_{[i]}}(X_{[i]}) - \varepsilon_{a_{i-[i]}}(X_{i-[i]})\right).$$

The polynomial $f_{\mathbf{a}}$ has the property that any monomial it contains has as most $|S_k| = 2^k-k-1$ distinct indeterminates X_j in it. For each $f_{\mathbf{a}}$, we construct a new polynomial as follows: for each variable X_j appearing in $f_{\mathbf{a}}$ with an exponent $e > 2^{k'}-1$, we replace X_j^e by X_j^e reduced modulo $X_j(X_j-1)\ldots(X_j-(2^{k'}-1))$. Call $f'_{\mathbf{a}}$ the new polynomial. Notice that, as functions over the rationals, $f_{\mathbf{a}}$ and $f'_{\mathbf{a}}$ have the same values over $\mathscr{A}$. As a function, each $f'_{\mathbf{a}}$ is in the span of all the $\sum_{i=0}^{2^k-k-1}(2^{k'}-1)^i\binom{2^k-1}{i}$ monomial functions in which at most 2^k-k-1 distinct variables enter and such that the exponent of each variable is at most $2^{k'}-1$.

From the hypothesis on A, Lemma 1 implies that the following holds for all $\mathbf{a}$ and $\mathbf{b}$ in A.

$$f'_{\mathbf{a}}(\mathbf{b}) = f_{\mathbf{a}}(\mathbf{b}) \equiv \begin{cases} 1 \bmod 2 & \text{if } \mathbf{a} = \mathbf{b} \\ 0 \bmod 2 & \text{if } \mathbf{a} \neq \mathbf{b} \end{cases}$$

We now show that this implies that the $|A|$ functions $f'_{\mathbf{a}}$ for $\mathbf{a} \in A$ are linearly independent over the rationals. Take $|A|$ rationals $\lambda_{\mathbf{a}}$ such that $\sum_{\mathbf{a}\in A}\lambda_{\mathbf{a}} f'_{\mathbf{a}} = \mathbf{0}$. Without loss of generality, we can actually consider that the $\lambda_{\mathbf{a}}$ are integers. The evaluation of the two sides of this expression at the point $\mathbf{b}$ gives $\lambda_{\mathbf{b}} \equiv 0 \bmod 2$. Thus, necessarily, $\lambda_{\mathbf{a}} \equiv 0 \bmod 2$ for all $\mathbf{a} \in A$. Suppose that the $\lambda_{\mathbf{a}}$ are not all zero and denote $\Lambda_i = \{\mathbf{a} \in A \text{ such that } \lambda_{\mathbf{a}} \neq 0 \text{ and } 2^i|\lambda_{\mathbf{a}}\}$ for i ranging from 1 to r, where r is the greatest integer such that 2^r appears in the prime power decomposition of some $\lambda_{\mathbf{a}}$. Evaluating, for increasing i from 1 to r, the functions $\sum_{\mathbf{a}\in\Lambda_i}(\lambda_{\mathbf{a}}/2^i)f'_{\mathbf{a}}$ gives that $\Lambda_1 = \emptyset$. Thus $\lambda_{\mathbf{a}} = 0$ for all $\mathbf{a} \in A$.

The fact that the $|A|$ functions $f'_{\mathbf{a}}$ are linearly independent over the rationals implies that

$$|A| \leq \sum_{i=0}^{2^k-k-1}(2^{k'}-1)^i\binom{2^k-1}{i} \leq \sum_{i=0}^{2^k-k}(2^{k'})^i\binom{2^k}{i}.$$

Now notice that, in the case $k' \geq k$, the function $h : j \mapsto (2^{k'})^j \binom{2^k}{j}$ is an increasing function over $\{0, \cdots, 2^k\}$: for any $i \in \{0, \cdots, 2^k - 1\}$, we have $h(i+1)/h(i) = 2^{k'}(2^k - i)/(i+1) \geq 2^{k'} 2^{-k} \geq 1$. We can now give the following upper bound.

$$|A| \leq 2^k \cdot (2^{k'})^{2^k - k} \binom{2^k}{2^k - k} = 2^k \cdot (2^{k'})^{2^k - k} \binom{2^k}{k}$$

Using the standard fact $\binom{2^k}{k} \leq (e2^k/k)^k$, where e is the Euler constant, we obtain, for $k \geq 3$,

$$|A| \leq 2^k (2^{k'})^{2^k - k} \left(2^k\right)^k = 2^{k' 2^k - k(k' - k - 1)},$$

which concludes the proof of the theorem. □

We are now ready to prove the lower bound on the classical nondeterministic complexity of $HEQ_{k,k'}$.

Theorem 3. *Let k and k' be two positive integers such that $k \geq 3$ and $k' \geq k$. Then $N(HEQ_{k,k'}) \geq k(k' - k) - (k + k')$.*

Proof. Denote again $\mathscr{A} = \{0, \cdots, 2^{k'} - 1\}^{2^k - 1}$. Notice that for any $\mathbf{a} \in \mathscr{A}$, $(\mathbf{a}, \mathbf{a})$ is a 1-instance of $HEQ_{k,k'}$. We show a lower bound on the number of 1-monochromatic (for $HEQ_{k,k'}$) rectangles of $\mathscr{A} \times \mathscr{A}$ necessary to cover $\{(\mathbf{a}, \mathbf{a}) | \mathbf{a} \in \mathscr{A}\}$. Here covering means that the union of the rectangles has only to include $\{(\mathbf{a}, \mathbf{a}) \mid \mathbf{a} \in \mathscr{A}\}$. Such a lower bound obviously implies a lower bound on the number of 1-monochromatic rectangles necessary to cover all the 1-instances of $HEQ_{k,k'}$.

Any 1-monochromatic rectangle of a covering of $\{(\mathbf{a}, \mathbf{a}) | \mathbf{a} \in \mathscr{A}\}$ can be considered, without loss of generality, to be of the form $A \times A$ for some subset $A \subseteq \mathscr{A}$. By the definition of a 1-monochromatic rectangle, for each $\mathbf{a} = (a_1, \ldots, a_{2^k - 1})$ and $\mathbf{b} = (b_1, \ldots, b_{2^k - 1})$ in A, Equation (1) must hold. Then, from Theorem 2, even for the largest 1-monochromatic rectangle of the form $A \times A$, we have $|A| \leq 2^{k' 2^k - k(k' - k - 1)}$. This implies that at least

$$\frac{(2^{k'})^{2^k - 1}}{|A|} \geq 2^{kk' - k^2 - k - k'}$$

1-monochromatic rectangles are necessary to cover $\{(\mathbf{a}, \mathbf{a}) \mid \mathbf{a} \in \mathscr{A}\}$. The nondeterministic complexity of $HEQ_{k,k'}$ is thus, using Fact 1, at least $kk' - k^2 - k - k'$. □

This theorem directly implies the quadratic separation.

Corollary 1. *There is a quadratic separation of quantum weakly nondeterministic and classical nondeterministic communication complexity.*

Proof. By considering for example the case $k' = 2k$. The quantum weakly nondeterministic communication complexity of $HEQ_{k,2k}$ is, from Theorem1, $O(k)$ and its classical nondeterministic communication complexity is, from Theorem 3, $\Omega(k^2)$. □

5 Open Problems

The main open problem is whether a separation of classical nondeterministic and quantum weakly nondeterministic communication complexity larger than quadratic can be shown between for a total function. Is an exponential gap achievable? It may indeed be the case that, for total functions, the largest gap achievable is polynomial and, possibly, quadratic.

Another interesting question is whether quantum proofs can be more helpful than classical proofs as far as quantum weakly nondeterminism is concerned, that is, when the proof has to be checked without error.

Acknowledgments

The author is grateful to H. Imai, T. Ito, H. Kobayashi and S. Tani for helpful comments about this work.

References

1. S. Aaronson. *Quantum Certificate Complexity.* Proceedings of 18th Annual IEEE Conference on Computational Complexity, pp. 171–178, 2003.
2. S. Aaronson and A. Ambainis. *Quantum Search of Spatial Regions.* Theory of Computing, 1, pp. 47–79, 2005.
3. L.M. Adleman, J. DeMarrais and M.A. Huang. *Quantum Computability.* SIAM Journal on Computing, 26(5), pp. 1524–1540, 1997.
4. L. Babai, H. Snevily and R. M. Wilson. *A New Proof of Several Inequalities on Codes and Sets.* Journal of Combinatorial Theory, Series A 71, pp. 146–153, 1995.
5. Z. Bar-Yossef, T. S. Jayram and I. Kerenidis. *Exponential Separation of Quantum and Classical One-way Communication Complexity.* Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pp. 128–137, 2004.
6. E. Bernstein and U. Vazirani. *Quantum Complexity Theory.* SIAM Journal on Computing, 26(5), pp. 1411–1473, 1997.
7. H. Buhrman. *Quantum Computing and Communication Complexity.* Bulletin of the EATCS, pp. 131–141, 2000.
8. H. Buhrman, R. Cleve and A. Wigderson. *Quantum vs. Classical Communication and Computation.* Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp. 63–68, 1998.
9. H. Buhrman, L. Fortnow, I. Newman and H. Röhrig. *Quantum Property Testing.* Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 480–488, 2003.
10. P. Delsarte. *An Algebraic Approach to the Association Schemes of Coding Theory.* Philips Res. Suppl. 10, 1973.
11. P. Delsarte. *The Association Schemes of Coding Theory.* In "Combinatorics; Proceedings of the NATO Advanced Study Institute, Breukelen, 1974, Part 1", Math. Centre Tracts, No. 55, Math. Centrum, Amsterdam, pp. 139–157, 1974.
12. D. Deutsch and R. Jozsa. *Rapid Solution of Problems by Quantum Computation.* Proceedings of the Royal Society of London Series A, 439, pp. 553–558, 1992.

13. S. Fenner, F. Green, S. Homer and R. Pruim. *Determining Acceptance Probability for a Quantum Computation is Hard for the Polynomial Hierarchy*. Proceedings of the Royal Society of London Series A, 455, pp. 3953–3966, 1999.
14. L. Fortnow and J. Rogers. *Complexity Limitations on Quantum Computation*. Journal of Computern and System Sciences, 59(2), pp. 240–252, 1999.
15. P. Frankl. *Orthogonal Vectors in the n-dimensional Cube and Codes with Missing Distance*. Combinatorica 6, pp. 279–285, 1986.
16. P. Høyer and R. de Wolf. *Improved Quantum Communication Complexity Bounds for Disjointness and Equality*. Proceedings of the 19th International Symposium of Theoretical Aspects of Computer Science, pp. 299–310, 2002.
17. A. Yu Kitaev. *Quantum NP*. Public Talk at the 2nd Workshop on Algorithms in Quantum Information Processing, 1999.
18. H. Klauck. *On Quantum and Probabilistic Communication: Las Vegas and One-way protocols*. Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, pp. 644–651, 2000.
19. H. Klauck. *Quantum Communication Complexity*. Proceedings of the Workshop on Boolean Functions and Applications at the 27th International Colloquium on Automata, Languages and Programming, pp. 241–252, 2000.
20. J. Hromkovič. *Communication Complexity and Parallel Computation*. Springer-Verlag, 1997.
21. E. Knill. *Quantum Randomness and Nondeterminism*. Los-Alamos e-print archive, quant-ph/9610012, 1996.
22. S. Massar, D. Bacon, N. Cerf and R. Cleve. *Classical Simulation of Quantum Entanglement without Local Hidden Variables*. Physics Reviews A, 63, 052305, 2001.
23. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
24. R. Raz. *Exponential Separation of Quantum and Classical Communication Complexity*. Proceedings of the 31st Annual ACM Symposium on Theory of Computing, pp. 358–367, 1999.
25. R. Raz and A. Shpilka. *On the Power of Quantum Proofs*. Proceedings of 19th Annual IEEE Conference on Computational Complexity, pp. 260–274, 2004.
26. J. Watrous. *Succint Quantum Proofs for Properties of Finite Groups*. Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pp. 537–546, 2000.
27. R. de Wolf. *Quantum Communication and Complexity*. Theoretical Computer Science, 287(1), pp. 337–353, 2002.
28. R. de Wolf. *Nondeterministic Quantum Query and Communication Complexity*. SIAM Journal on Computing 32(3), pp. 681–699, 2003.
29. T. Yamakami and A. C. -C. Yao. $NQP_C = co - C_=P$. Information Processing Letters 71, pp. 63–69, 1999.

Minimal Chordal Sense of Direction and Circulant Graphs

Rodrigo S.C. Leão and Valmir C. Barbosa*

Universidade Federal do Rio de Janeiro
Programa de Engenharia de Sistemas e Computação, COPPE
Caixa Postal 68511, 21941-972 Rio de Janeiro - RJ, Brazil
rleao@cos.ufrj.br, valmir@cos.ufrj.br

Abstract. A sense of direction is an edge labeling on graphs that follows a globally consistent scheme and is known to considerably reduce the complexity of several distributed problems. In this paper we study a particular instance of sense of direction, called a chordal sense of direction (CSD). In special, we analyze the class of k-regular graphs that admit a CSD with exactly k labels (a minimal CSD). We prove that connected graphs in this class are Hamiltonian and that the class is equivalent to that of circulant graphs, presenting an efficient (polynomial-time) way of recognizing it when the graphs' degree k is fixed.

Keywords: Chordal sense of direction; Cayley graphs; Circulant graphs.

1 Introduction

In this paper we model a distributed system as an undirected graph G on n vertices having no multiple edges or self-loops. Vertices according to this model stand for processors, edges for bidirectional communication channels. For terminology or notation on graph theory not defined here we refer the reader to [1].

Every edge of G is assumed to have two labels, each corresponding to one of its end vertices. In [2], a property of this edge labeling was introduced which can considerably reduce the complexity of many problems in distributed computing [3]. This property refers to the ability of a vertex to distinguish among its incident edges according to a globally consistent scheme and is formally described in [4]. An edge labeling for which the property holds is called a *sense of direction* and is necessarily such that all the edge labels corresponding to a same vertex are distinct (the edge labeling is then what is called a *local orientation*). We say that a sense of direction is *symmetric* if it is possible to derive the label corresponding to one end vertex of an edge from the label corresponding to the other. We say that it is *minimal* if it requires exactly $\Delta(G)$ distinct labels, where $\Delta(G)$ is the maximum degree in G. For a survey on sense of direction, we refer the reader to [5].

* Corresponding author.

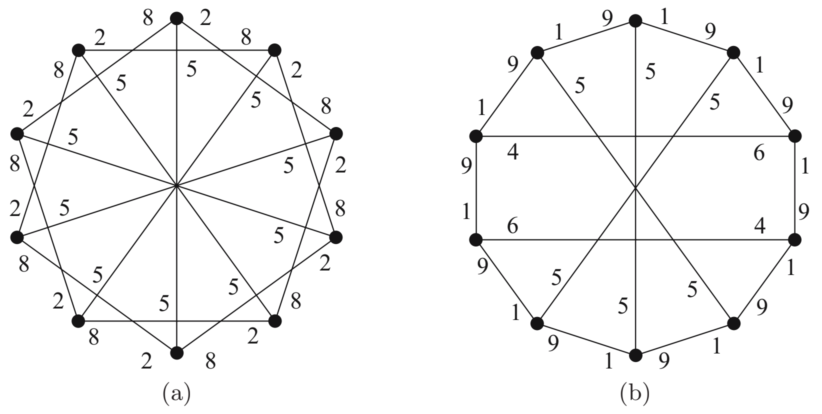


Fig. 1. A graph with an edge labeling that is an MCSD (a) and another graph with an edge labeling that is a CSD but not an MCSD (b). Vertices are ordered clockwise.

A particular instance of symmetric sense of direction, called a *chordal sense of direction* (CSD), can be constructed on any graph by fixing an arbitrary cyclic ordering of the vertices and, for each edge uv, selecting the difference (modulo n) from the rank of u in the ordering to that of v as the label of uv that corresponds to u (likewise, the label that corresponds to v is the rank difference from v to u). In Figure 1(a), an example is given of a minimal chordal sense of direction (MCSD). It is relatively easy to see that there exist graphs that do not admit an MCSD, as for instance the one in Figure 1(b).

Given a finite group A and a set of generators $S \subseteq A$, a *Cayley graph* is a graph H whose vertices are the elements of the group ($V(H) = A$) and whose edges correspond to the action of the generators ($uv \in E(H) \iff \exists s \in S : v = s * u$, where $*$ is the operation defined for A). We assume that the set of generators is closed under inversion, so H is an undirected graph. An edge labeling of H assigning two labels to each edge in such a way that each of an edge's labels corresponds to one of its end vertices is called a *Cayley labeling* if, for edge uv, the label that corresponds to vertex u is s such that $v = s * u$.

In [6], it was shown that a regular graph's edge labeling is a minimal symmetric sense of direction if and only if the graph is a Cayley graph and the labeling is a Cayley labeling. This result was later extended to directed graphs in [7], where the problem of recognizing labeled Cayley graphs was also demonstrated to be solvable in parallel polynomial time. This latter result uses the same $O(n^{14.256})$-time algorithm of [8], where the problem of deciding whether a given labeling is a sense of direction of a given graph was solved.

A *circulant graph* (also known as a *chordal ring*) is a Cayley graph over $\mathbb{Z}_n$, the cyclic group of order n under the addition operation. The relevance of circulant graphs is due to their connectivity properties (small diameter, high symmetry, etc.), which render them excellent topologies for network interconnection, VLSI, and distributed systems [9]. The problem of recognizing circulant graphs is still challenging: results are known only for very specific instances, like the cases of n prime [10], geometric circulant graphs [11], and recursive circulant graphs [12].

In this paper we analyze the regular graphs that admit an MCSD. We describe their structure, show that they are all Hamiltonian if connected, and moreover demonstrate an equivalence between certain distinct labelings. We also show that the class of regular graphs that admit an MCSD and the class of circulant graphs are equivalent to each other. A straightforward consequence of our analysis is that the problem of recognizing circulant graphs can be polynomially solved when the graphs' degree is fixed.

The remainder of the paper is organized in the following manner. We start in Section 2 with basic results on the structure of regular graphs that admit an MCSD. Then in Section 3 we develop the argument that all such graphs, if connected, are necessarily Hamiltonian. In Section 4 we discuss the problem of deciding whether a k-regular graph admits an MCSD and show, for fixed k, that this can be achieved polynomially. This result carries on to the recognition of circulant graphs after we prove, in Section 5, the equivalence of such graphs and regular graphs that admit an MCSD. Conclusions follow in Section 6.

Throughout the text, the operators $+_n$, $-_n$, and $\cdot_n$ represent, respectively, the modulo-n operations of addition, subtraction, and multiplication.

2 MCSD's of Regular Graphs

Let G be a k-regular graph that admits an MCSD, λ a labeling that is an MCSD of G, and $\Gamma \subseteq \{1,\ldots,n-1\}$ the set of labels used by λ. Since λ is minimal, we may write $\Gamma = \{\gamma_1,\ldots,\gamma_k\}$ and assume, further, that $\gamma_1 < \cdots < \gamma_k$. We denote by $\lambda_u(uv)$ the label of edge uv that corresponds to vertex u. We also write $\lambda(uv) = \{\lambda_u(uv), \lambda_v(uv)\} = \{\gamma_i, \gamma_j\}$ with $1 \leq i, j \leq k$. It is easy to see that for any CSD there exists a symmetry function ψ such that $\psi(\lambda_u(uv)) = \lambda_v(uv)$, given by $\psi(\gamma_i) = n - \gamma_i$ for $\gamma_i \in \Gamma$. We start by highlighting an important property of λ.

Lemma 1. *If k is even, then the edges of G are labeled by λ with the label pairs $\{\gamma_1, n-\gamma_1\},\ldots,\{\gamma_{k/2}, n-\gamma_{k/2}\}$. If k is odd, then a further label pair is $\{\gamma_{\lceil k/2\rceil}, n-\gamma_{\lceil k/2\rceil}\} = \{n/2, n/2\}$.*

Proof. The $k/2$ label pairs for the case of k even follow directly from the definition of ψ and from the fact that $|\Gamma| = k$. If k is odd, then the label $\gamma_{\lceil k/2\rceil} = n/2$ (necessarily an integer, since k odd implies n even when G is k-regular) remains unused by any of those pairs, so a further label pair is $\{n/2, n/2\}$. □

By Lemma 1, we can always refer to λ by simply giving the $\lceil k/2\rceil$ labels that are no greater than $n/2$. Having established this property of λ, we now set out to describe more about the structure of graphs that admit an MCSD. In what follows, we say that a graph H *decomposes* into the two subgraphs A and B when $V(A) \cup V(B) = V(H)$, $E(A) \cup E(B) = E(H)$, and $E(A) \cap E(B) = \emptyset$. Also, recall that a 2-factor of H is a collection of vertex-disjoint cycles from H that spans all of its vertices.

Theorem 2. *G decomposes into $\lfloor k/2 \rfloor$ 2-factors and, if k is odd, a perfect matching as well. For $1 \leq i \leq \lfloor k/2 \rfloor$, the edges of the ith 2-factor are labeled with γ_i, and the edges of the perfect matching with $n/2$.*

Proof. The k edges incident to each vertex are labeled with distinct members of Γ on their near ends. So each of the $\lfloor k/2 \rfloor$ label pairs asserted initially in Lemma 1 can be used to identify a different 2-factor. Such 2-factors encompass all of G, with the exception of the edges whose label pair is $\{n/2, n/2\}$ in the odd-k case (again, in Lemma 1). But these clearly constitute a perfect matching in G. □

3 The Necessity of a Hamiltonian Cycle

In this section we assume that G is connected and begin by asserting a relationship between two vertices that belong to a same cycle of one of the 2-factors established in Theorem 2. Let us denote by $r(u)$ the rank of vertex u in the cyclic ordering that underlies the CSD.

Lemma 3. *For $1 \leq i \leq \lfloor k/2 \rfloor$, two vertices u and v belong to a common cycle of the 2-factor whose edges are labeled with γ_i if and only if $r(v) = r(u) +_n t\gamma_i$ for some integer $t \geq 0$.*

Proof. If u and v share a common γ_i-labeled cycle, then traversing the cycle from u to v in the direction that exits a vertex along the γ_i-labeled end of the edge adds (modulo n) γ_i rank units to $r(u)$ for each edge traversed. Then there exists a nonnegative integer t such that $r(v) = r(u) +_n t\gamma_i$.

Conversely, if $r(v) = r(u) +_n t\gamma_i$ for some $t \geq 0$, then, since every vertex has an incident edge labeled with γ_i on the near end, v can be reached by a path that begins at u, exclusively uses edges labeled with γ_i, and has t edges. Clearly, such a path is part of a cycle of a 2-factor whose edges are labeled with γ_i. □

Recall now that two integers a and b are *relative primes*, denoted by $a \perp b$, if $\gcd(a, b) = 1$.

Fact 4. *Let a and b be integers. Then $a \perp n$ if and only if the smallest $b > 0$ that satisfies $b \cdot_n a = 0$ is $b = n$.*

We are now in position to demonstrate that G is Hamiltonian. We do this by splitting the proof into cases that bear on the relative primality between each of $\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}$ and n.

Theorem 5. *If there exists $\gamma_i \in \{\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}\}$ such that $\gamma_i \perp n$, then G has a Hamiltonian cycle whose edges are labeled with γ_i.*

Proof. By Fact 4, the smallest integer $t > 0$ that satisfies $t \cdot_n \gamma_i = 0$ is $t = n$. In the same way, for any vertex u, the smallest $t > 0$ that satisfies $r(u) +_n t\gamma_i = r(u)$ is also $t = n$. By Lemma 3, vertex u is on an n-vertex cycle whose edges are labeled with γ_i. □

Before we proceed to the case in which no $\gamma_i \in \{\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}\}$ is such that $\gamma_i \perp n$, we give a necessary condition for such a scenario to happen.[1] We also prove a general property on the relative primality between the greatest common divisor of all the labels that are no greater than $n/2$ (including, if applicable, $n/2$ itself) and n.

Lemma 6. *If no $\gamma_i \in \{\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}\}$ exists such that $\gamma_i \perp n$, then for $1 \leq i \leq \lfloor k/2 \rfloor$ the edges of G labeled with γ_i form a 2-factor with d_i cycles of the same length, where $d_i = \gcd(\gamma_i, n) > 1$.*

Proof. First rewrite $t \cdot_n \gamma_i = 0$ as $t \cdot_{n/d_i} \gamma_i/d_i = 0$ and notice that $\gamma_i/d_i \perp n/d_i$. By Fact 4, it follows that $t = n/d_i$ is the smallest integer that satisfies $t \cdot_{n/d_i} \gamma_i/d_i = 0$, and hence also $t \cdot_n \gamma_i = 0$. By Lemma 3, each cycle of the 2-factor whose edges are labeled with γ_i comprises n/d_i vertices, so by Theorem 2 the number of such cycles is d_i. $\square$

Lemma 7. *Let $d = \gcd(\gamma_1, \ldots, \gamma_{\lceil k/2 \rceil})$. Then $d \perp n$.*

Proof. Let $t_i \geq 0$, for $1 \leq i \leq \lceil k/2 \rceil$, be an integer. Thus, $\sum_i t_i \gamma_i$ is a multiple of d and represents, for an arbitrary walk in G, the rank difference along the CSD's cyclic vertex ordering from the walk's initial vertex to its final vertex. To see why any walk is thus contemplated, notice that a walk that uses t_i edges labeled with $n - \gamma_i$ on their near ends as vertices are exited along the walk can be substituted for by another one that connects the same two vertices and uses $n/\gcd(\gamma_i, n) - t_i$ edges labeled with γ_i instead. Therefore, for a walk between arbitrary vertices u and v, the rank difference between these vertices in the cyclic ordering is given by $t \cdot_n d$ for some $t \in \{0, \ldots, n-1\}$. And, since G is connected, every possible value of $t \cdot_n d$ (i.e., $0, \ldots, n-1$) must result from a distinct value of t. If such is the case, then $\gcd(d, n) = 1$, that is, $d \perp n$.[2] $\square$

It is important to recall that, when k is odd, $\gamma_{\lceil k/2 \rceil} = n/2$ by Lemma 1. In this case, by Lemma 7 we must have $d = 1$ (since d divides $n/2$, therefore n as well, which for $d > 1$ contradicts the lemma). The existence of a Hamiltonian cycle when none of $\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}$ is relatively prime to n can now be proven.

Theorem 8. *If no $\gamma_i \in \{\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}\}$ exists such that $\gamma_i \perp n$, then G has a Hamiltonian cycle.*

Proof. For γ_i any member of $\{\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}\}$, let $C_1, \ldots, C_{d_i}$ be the cycles of the 2-factor whose edges are labeled with γ_i. By Lemma 6, $d_i = \gcd(\gamma_i, n)$; by hypothesis, $d_i > 1$. We show that a Hamiltonian cycle exists that uses portions of the cycles $C_1, \ldots, C_{d_i}$ and interconnects them by edges with labels other than γ_i.

[1] The reader should note that Theorem 5 and Lemma 6 could be coalesced into one single result stating that the number of cycles in the 2-factor whose edges are labeled with γ_i for $1 \leq i \leq \lfloor k/2 \rfloor$ is $\gcd(\gamma_i, n)$. We choose to do otherwise for clarity's sake only.

[2] A formal proof of this implication can be found in Section 4.8 of [13].

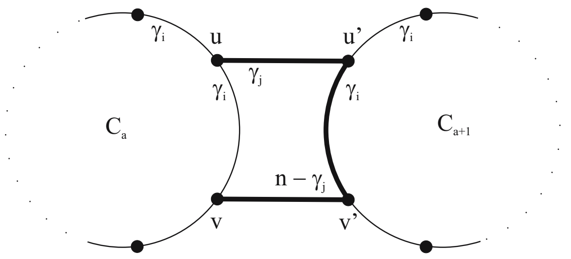


Fig. 2. A path connecting vertices u and v and whose edges are labeled with γ_j, γ_i, and $n - \gamma_j$ (in this order). Cycles C_a and C_{a+1} belong to the 2-factor whose edges are labeled with γ_i.

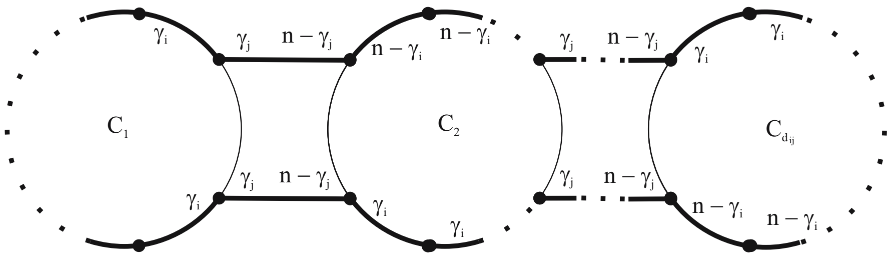


Fig. 3. Larger cycle using the cycles whose edges are labeled with γ_i and edges labeled with γ_jL

Let $\gamma_j \in \{\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}\}$ be such that it is not a multiple of d_i.[3] If u and v are adjacent vertices on a cycle C_a in $\{C_1, \ldots, C_{d_i - 1}\}$, then there exist edges labeled with γ_j that connect u and v to vertices u' and v', respectively, where u' and v' are adjacent on cycle C_{a+1}. This is summarized in the equality

$$\gamma_j +_n \gamma_i +_n (n - \gamma_j) = \gamma_i, \tag{1}$$

which refers to the illustration in Figure 2.

If we let $d_{ij} = \gcd(\gamma_i, \gamma_j)$, we can then interconnect groups of d_{ij} cycles from $\{C_1, \ldots, C_{d_i}\}$ through the edges labeled with γ_j, as in Figure 3, yielding d_i/d_{ij} larger cycles. If $\gamma_i \perp \gamma_j$ (i.e., $d_{ij} = 1$), then the proof is complete. Otherwise, we can interconnect these larger cycles in the same way through edges labeled with some γ_k such that $d_k \perp d_j$.

The case in which none of $\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}$ is a γ_k such that $d_k \perp d_j$ has $d' = \gcd(\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}) > 1$. In this case, if no edges are labeled with $n/2$, then every path in G, say from u to v, is such that $r(v) -_n r(u)$ is a multiple (modulo n) of d'. And since G is connected, this has to hold for all vertex pairs in the graph, even those whose rank differences are not a multiple of d'. This is clearly contradictory, so there have to exist edges labeled with $n/2$ (in which case k

[3] The case in which every one of $\gamma_1, \ldots, \gamma_{\lfloor k/2 \rfloor}$ is a multiple of d_i has a trivial Hamiltonian cycle that uses only edges labeled with γ_i and $n/2$.

must be odd, by Lemma 1) and we must have $d' \perp n/2$ (by Lemma 7, according to which $\gcd(d', n/2) \perp n$). This latter conclusion allows us to substitute $n/2$ for γ_j in (1), and then we see that the edges labeled with $n/2$ connect two distinct cycles of the 2-factor whose edges are labeled with γ_i. Thus, the larger cycles formed by interconnecting the cycles $C_1, \ldots, C_{d_i}$ through edges labeled with γ_j can be appropriately interconnected by the edges labeled with $n/2$. □

The following is then straightforward.

Corollary 9. *G has a Hamiltonian cycle.*

Proof. By Theorems 5 and 8. □

Note, finally, that even though the presence of a Hamiltonian cycle is a necessary condition for an edge labeling to be an MCSD in connected regular graphs, it is not a sufficient condition. In fact, it is easy to find k-regular graphs that have a Hamiltonian cycle but do not admit a CSD with k labels, as the example in Figure 1(b).

4 Deciding Whether an MCSD Exists

Given an arbitrary set $\{\gamma_1, \ldots, \gamma_{\lceil k/2 \rceil}\}$ of labels such that $\gamma_1 < \cdots < \gamma_{\lceil k/2 \rceil}$, if none of its members is greater than $n/2$ with $\gamma_{\lceil k/2 \rceil} = n/2$ in the odd-k case, then one can easily (polynomially) generate a k-regular graph H with an MCSD by simply arranging the n vertices in a cyclic ordering and, for each $\gamma_i \in \{\gamma_1, \ldots, \gamma_{\lceil k/2 \rceil}\}$, connecting pairs of vertices whose ranks in the ordering differ by γ_i and labeling the resulting edges with $\{\gamma_i, n - \gamma_i\}$ appropriately. In order for H to be connected, by Lemma 7 we require in addition that $\gcd(\gamma_1, \ldots, \gamma_{\lceil k/2 \rceil}) \perp n$. Thus, a possible direction towards the development of an algorithm to check whether a given k-regular graph G admits an MCSD is to generate H in this way for every pertinent set of labels,[4] and then to check whether H is isomorphic to G.

When we fix the input graph's degree (i.e., k is a constant), the maximum number of candidate labelings to be checked if we ignore the restriction that $\gcd(\gamma_1, \ldots, \gamma_{\lceil k/2 \rceil}) \perp n$ in the connected case is

$$\binom{\lfloor n/2 \rfloor}{\lfloor k/2 \rfloor} = O(n^k),$$

a polynomial in n. In [14], the isomorphism of graphs of bounded degree was shown to be testable in polynomial time. Thus, we can decide whether G admits an MCSD also polynomially.

The complexity of the overall MCSD test can be clearly improved if we consider the possible isomorphism between graphs generated from distinct valid label sets. We say that two distinct labelings λ and λ' are *equivalent*, denoted

[4] It is easy to see that H is unique for a given set of labels.

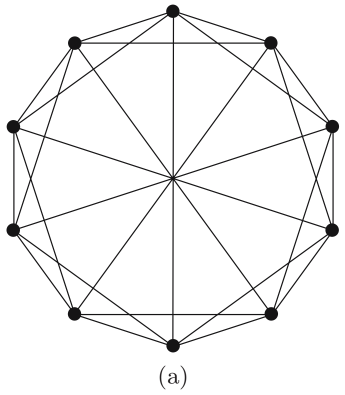
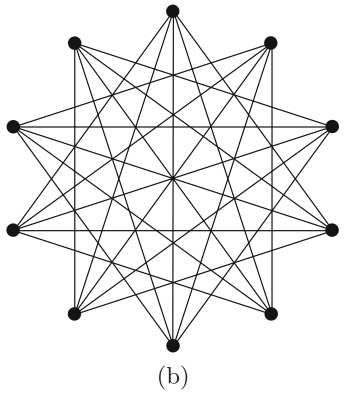



Fig. 4. Isomorphic graphs with equivalent MCSD labelings, based on the label sets $\{1, 2, 5\}$ (a) and $\{3, 4, 5\}$ (b)

by $\lambda \equiv \lambda'$, if they generate isomorphic graphs. For example, it can be easily seen that the labelings λ and λ', drawing respectively on the label sets $\{1, 2, 5\}$ and $\{3, 4, 5\}$, generate isomorphic 5-regular graphs on 10 vertices (see Figure 4), so $\lambda \equiv \lambda'$. Let us then consider a transformation of λ into λ' that preserves the MCSD property. In what follows, λ draws on the set $\{\gamma_1, \dots, \gamma_{\lceil k/2 \rceil}\}$ for labels, λ' on $\{\gamma'_1, \dots, \gamma'_{\lceil k/2 \rceil}\}$.

Theorem 10. *Let $\alpha < n/2$ be an integer such that $\alpha \perp n$. For $1 \le i \le \lceil k/2 \rceil$, let*

$$\gamma'_i = \begin{cases} \alpha \cdot_n \gamma_i & \text{if } \alpha \cdot_n \gamma_i \le n/2 \\ n - \alpha \cdot_n \gamma_i & \text{if } \alpha \cdot_n \gamma_i > n/2. \end{cases}$$

If λ is an MCSD for G, then so is λ'.

Proof. It suffices that we argue that no member of $\{\gamma'_1, \dots, \gamma'_{\lceil k/2 \rceil}\}$ is greater than $n/2$ with $\gamma_{\lceil k/2 \rceil} = n/2$ for k odd, that every two members of this set are distinct, and also that the vertices can be rearranged cyclically so that λ' is indeed a CSD. The first of these properties holds trivially and the second follows from well-known number-theoretic properties.[5] As for the third property, clearly it suffices for the vertices to be arranged into a cyclic ordering in which vertex u has rank $r'(u)$ such that $r'(u) = \alpha \cdot_n r(u)$. $\square$

It is easy to see that any $\alpha > n/2$ would produce the same results as its symmetric modulo n. We can also see that any λ comprising a γ_i such that $\gamma_i \perp n$ can yield a λ' with $\gamma'_i = 1$. For such, it is sufficient to take α as the multiplicative inverse (modulo n) of γ_i.

It is also curious to note that the transformation in Theorem 10 ensures that $\gamma_i \perp n$ if and only if $\gamma'_i \perp n$. To see this, consider for example the case of $\gamma'_i = \alpha \cdot_n \gamma_i$. By using Euclid's Theorem [13] and the fact that $\alpha \perp n$ in succession, we have $\gcd(\alpha \cdot_n \gamma_i, n) = \gcd(\alpha\gamma_i, n) = \gcd(\gamma_i, n)$, thence $\gcd(\gamma'_i, n) = \gcd(\gamma_i, n)$.

[5] We once again refer the reader to Section 4.8 of [13].

5 MCSD's and Circulant Graphs

There is a clear equivalence between circulant graphs and regular graphs that admit an MCSD. We describe it formally in the following theorem.

Theorem 11. *G is circulant of generator set S if and only if it is $|S|$-regular and admits an MCSD.*

Proof. Let G be a circulant graph of generator set S. Then $uv \in E(G)$ if and only if there exists $s \in S$ such that $v = u +_n s$. Let λ be a labeling for G such that $\lambda_u(uv) = s$. Since the vertices of G are elements of $\mathbb{Z}_n$, they already have a natural cyclic ordering in which $r(u) = u$ for all $u \in V(G)$. So $\lambda_u(uv) = r(v) -_n r(u)$ and λ is a CSD of G. Also, as every vertex is connected to the vertex ranking s higher (modulo n) than itself for every $s \in S$, G is $|S|$-regular and λ uses $|S|$ labels (thence the CSD is minimal).

Conversely, if G is a k-regular graph that admits an MCSD, then $|\Gamma| = k$ and there exists a cyclic ordering of the vertices such that each edge uv is labeled with $\lambda_u(uv) = r(v) -_n r(u)$, where $0 \leq r(u) \leq n-1$. Letting $V(G) = \mathbb{Z}_n$ so that $u = r(u)$ and $S = \Gamma$ yields $\lambda_u(uv) = v -_n u$ for all $uv \in E(G)$, thence $v = u +_n \lambda_u(uv)$. G is therefore circulant of generator set S. □

One first example of how Theorem 11 sheds new light on the two concepts involved comes from considering the result on circulant graphs in [15], which implies in the connected case that $\gcd(s_0, \ldots, s_k, n) = 1$, where s_i, for $0 \leq i \leq k$, is an element of the set of generators. This is of course coherent with Lemma 7 and Corollary 9, and a straightforward implication of the general result of [15] is that a circulant graph is Hamiltonian if and only if it is connected [16]. Our approach introduces new ways of constructing Hamiltonian cycles in this case.

Another interesting insight is the following. An $n \times n$ matrix is said to be *circulant* if its ith line is the cyclic shift of the first line by i positions, where $0 \leq i \leq n-1$. Another characterization of circulant graphs is that a graph is circulant if its adjacency matrix is circulant. By the equivalence established in Theorem 11, it then becomes possible to approach the problem of recognizing regular graphs that admit an MCSD along a different route: since it is well-known that the isomorphism between two graphs G and H can be viewed as a permutation of lines and columns of the adjacency matrix of G ($A(G)$) that generates that of H, we can test whether a regular graph G admits an MCSD by finding a permutation of lines and columns of $A(G)$ such that the resulting matrix is circulant.

We note, in addition, that the transformation defined in Theorem 10 also has an analogue in the literature on circulant graphs. Let G and H be circulant graphs such that their sets of generators are R and S, respectively. We say that R and S are *proportional*, denoted by $R \sim S$, if, for some $a \perp n$, $r = a \cdot_n s$ bijectively for $r \in R$ and $s \in S$. Clearly, if $R \sim S$ then G is isomorphic to H. The converse statement was conjectured in [17] and is known as *Ádám's conjecture*. However, in [18] the conjecture was proven false.

The problem of recognizing circulant graphs, finally, is probably the one most affected by the equivalence of Theorem 11. Even though the algorithm we suggest to test whether a k-regular graph admits an MCSD is polynomial only for fixed k, when applied to the context of circulant graphs it is the only known result on arbitrary topologies (without any restrictions on the structure or the number of vertices) for that class.

6 Conclusions

We have in this paper considered two classes of graphs, namely those that are regular and admit an MCSD and those that are circulant. We have contributed new properties and insights to the study of graphs in both classes.

Our contribution to the study of regular graphs that admit an MCSD has been a detailed study of their structure in terms of 2-factors and perfect matchings, and also proving that, when connected, they are necessarily Hamiltonian. We have in addition shown that fixed-degree regular graphs can be tested for admitting an MCSD in polynomial time. This latter result is based on the well-known result that testing two bounded-degree graphs for isomorphism can be achieved polynomially.

After demonstrating that regular graphs with an MCSD are equivalent to circulant graphs, we then discussed several new insights into the study of circulant graphs that the equivalence highlights. Of special significance has been the discovery of the first result on the recognition of arbitrary circulant graphs: if they have fixed degree, then they can be recognized in polynomial time as well.

Acknowledgments

The authors acknowledge partial support from CNPq, CAPES, and a FAPERJ BBP grant.

References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. North-Holland, New York, NY (1976)
2. Santoro, N.: Sense of direction, topological awareness and communication complexity. SIGACT News **2** (1984) 50–56
3. Flocchini, P., Mans, B., Santoro, N.: On the impact of sense of direction on message complexity. Information Processing Letters **63** (1997) 23–31
4. Flocchini, P., Mans, B., Santoro, N.: Sense of direction: definitions, properties and classes. Networks **32** (1998) 165–180
5. Flocchini, P., Mans, B., Santoro, N.: Sense of direction in distributed computing. Theoretical Computer Science **291** (2003) 29–53
6. Flocchini, P.: Minimal sense of direction in regular networks. Information Processing Letters **61** (1997) 331–338
7. Boldi, P., Vigna, S.: Minimal sense of direction and decision problems for Cayley graphs. Information Processing Letters **64** (1997) 299–303

8. Boldi, P., Vigna, S.: On the complexity of deciding sense of direction. SIAM Journal on Computing **29** (2000) 779–789
9. Bermond, J.C., Cornellas, F., Hsu, D.F.: Distributed loop computer networks: a survey. Journal of Parallel and Distributed Computing **24** (1995) 2–10
10. Muzychuk, M., Tinhofer, G.: Recognizing circulant graphs of prime order in polynomial time. The Electronic Journal of Combinatorics **5** (1998) #R25
11. Muzychuk, M., Tinhofer, G.: Recognizing circulant graphs in polynomial time: an application of association schemes. The Electronic Journal of Combinatorics **8** (2001) #R26 in number 1
12. Fertin, G., Raspaud, A.: Recognizing recursive circulant graphs. Electronic Notes in Discrete Mathematics **5** (2000) 112–115
13. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics: A Foundation for Computer Science. Second edn. Addison-Wesley, Boston, MA (1994)
14. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of Computer and System Sciences **25** (1982) 42–65
15. Boesch, F., Tindell, R.: Circulants and their connectivities. Journal of Graph Theory **8** (1984) 487–499
16. Burkard, R.E., Sandholzer, W.: Efficiently solvable special cases of bottleneck travelling salesman problems. Discrete Applied Mathematics **32** (1991) 61–76
17. Ádám, A.: Research problem 2-10. Journal of Combinatorial Theory **2** (1967) 393
18. Elspas, B., Turner, J.: Graphs with circulant adjacency matrices. Journal of Combinatorial Theory **9** (1970) 297–307

Querying and Embedding Compressed Texts

Yury Lifshits[1] and Markus Lohrey[2]

[1] Steklov Institut of Mathematics, St.Petersburg, Russia
[2] Universität Stuttgart, FMI, Germany
yura@logic.pdmi.ras.ru, lohrey@informatik.uni-stuttgart.de

Abstract. The computational complexity of two simple string problems on compressed input strings is considered: the querying problem (What is the symbol at a given position in a given input string?) and the embedding problem (Can the first input string be embedded into the second input string?). Straight-line programs are used for text compression. It is shown that the querying problem becomes P-complete for compressed strings, while the embedding problem becomes hard for the complexity class Θ_2^p.

1 Introduction

During the last decade, the massive increase in the volume of data has motivated the need for algorithms on *compressed data*, like for instance compressed strings, trees, or pictures. The general goal is to develop efficient algorithms that directly work on compressed data without prior decompression, or to prove under general assumptions from complexity theory that such efficient algorithms do not exist. In this paper we concentrate on algorithms on compressed strings. We investigate two computational problems, which can be trivially solved in linear time for uncompressed input strings: the querying problem and the embedding problem. In the embedding problem we have given two input strings p (the pattern) and t (the text), and we ask whether p can be embedded into t, i.e., p can be obtained by deleting some letters of the text t at arbitrary positions, see Section 4 for a formal definition. In the querying problem the input consists of a string s, a position $i \in \mathbb{N}$, and a letter a, and we ask, whether the i-th symbol of s is a.

For string compression, we choose *straight-line programs* (SLPs), i.e., context-free grammars that generate exactly one word. Straight-line programs turned out to be a very flexible and mathematically clean compressed representation of strings. Several other dictionary-based compressed representations, like for instance Lempel-Ziv (LZ) factorizations [24], can be converted in polynomial time into straight-line programs and vice versa [18]. This implies that complexity results, which refer to classes above deterministic polynomial time, can be transfered from SLP-encoded input strings to LZ-encoded input strings. It turns out that the computational complexity of the querying problem and the embedding problem becomes very different, when input strings are encoded via SLPs: While for SLP-compressed strings the querying problem (also called *compressed querying problem*) becomes complete for deterministic polynomial time (Thm. 4), the embedding problem (also called *fully compressed embedding problem*; the term "fully" is used because both, the pattern and the text are assumed to be compressed) becomes hard for the class Θ_2^p (Thm. 1). The latter class consists of all problems that can be accepted by a deterministic polynomial time machine with access to an oracle from NP and such that furthermore all questions to the oracle are asked in parallel [23]. Θ_2^p is located between the first and the second level of the polynomial time hierarchy; it contains NP and coNP and is contained in $\Sigma_2^p \cap \Pi_2^p$. We are currently not able to prove a matching upper bound. The best upper bound for the fully compressed embedding problem that we can prove is PSPACE (Prop. 1). A corollary of the Θ_2^p-hardness of the fully compressed embedding problem is Θ_2^p-hardness of the *longest common subsequence problem* and the *shortest common supersequence problem* on SLP-compressed strings, even when both problems are restricted to two input strings. These problems have many applications e.g. in computational biology [10].

The paper is organized as follows. After introducing the necessary concepts in Sec. 2, we prove in Sec. 3, based on a reduction from the super increasing subset sum problem [11], P-completeness of the compressed querying problem for a binary input alphabet. For a variable input alphabet, we sharpen this result by showing that even for RLZ-encoded strings the compressed querying problem is P-complete, which solves an open problem from [7]. RLZ-encodings (restricted Lempel-Ziv encodings) can be seen as a restricted form of straight-line programs. In Sec. 4 we show that the fully compressed embedding problem is Θ_2^p-hard. The proof is divided into two main parts. First we prove NP-hardness by a reduction from the subset sum problem (Sec. 4.1). Second, we show how to simulate boolean operations via fully compressed embedding (Sec. 4.2). By taking together these two parts we can deduce hardness for Θ_2^p (Sec. 4.3).

Let us briefly discuss related work. Research on pattern matching problems for dictionary-based compressed strings started with the seminal paper [1]. In [17], a polynomial time algorithm for testing whether two SLPs represent the same text was presented. The technique of [17] was extended in [8,14] in order to show that the *fully compressed pattern matching problem* can be solved in polynomial time as well. The fully compressed pattern matching problem is the compressed version of the classical pattern matching problem: for two given SLPs P and T we ask, whether the text represented by T can be written as upv, where p is the text represented by the SLP P. Note the difference between the *fully compressed pattern matching problem* and *the fully compressed embedding problem* studied in this paper: In the latter problem we also search for a compressed pattern in a compressed text, but we allow that the pattern occurs scattered, i.e., with gaps, in the text. This more liberal notion of pattern-occurrence makes the application of periodicity properties of words, which are crucial in [8,14,17], impossible, and is in some sense the reason for the higher complexity of the fully compressed embedding problem. A similar complexity jump was observed when moving from ordinary (1-dimensional) to 2-dimensional texts, i.e., rectangular pictures: In this framework, fully compressed pattern matching becomes Σ_2^P-complete [3].

The computational problems mentioned so far can be all formulated as particular compressed membership problems, where we ask whether a given compressed text belongs to some formal language, which may either be fixed or given in the input, e.g., in form of an automaton or a grammar. Precise complexity results for these problems were obtained in [2,13] for regular languages and [12] for context-free languages.

Whereas it is NP-complete to compute (and even hard to approximate up to a constant factor) a minimal SLP that generates a given input string [4], several approaches for generating a small SLP that produces a given input string were proposed and analyzed in the literature, see e.g. [4,21].

We refer to [7,15,18,19,20,22] for a more detailed discussion of algorithmic problems on compressed strings.

2 Preliminaries

We assume that the reader has some basic background in complexity theory [16]. Let Σ be a finite alphabet. The *empty word* over Σ is denoted by ε. For a word $s = a_1 \cdots a_n \in \Sigma^*$ ($a_i \in \Sigma$) let $|s| = n$, $|s|_a = |\{i \mid a_i = a\}|$ (for $a \in \Sigma$), $s[i] = a_i$ (for $1 \leq i \leq n$), and $s[i,j] = a_i a_{i+1} \cdots a_j$ (for $1 \leq i \leq j \leq n$). If $i > j$ we set $s[i,j] = \varepsilon$.

Following [18], a *straight-line program (SLP) over the terminal alphabet Σ* is a context-free grammar G with ordered non-terminal symbols $X_1, \ldots, X_m$ (X_m is the starting symbol) such that there is exactly one production for each symbol: either $X_i \to a$, where $a \in \Sigma$ is a terminal, or $X_i \to X_j X_k$ for some $j, k < i$. The language generated by the SLP G contains exactly one word that is denoted by $\mathrm{eval}(G)$. More generally, every nonterminal X_i produces exactly one word that is denoted by $\mathrm{eval}_G(X_i)$. We omit the index G if the underlying SLP is clear from the context. The size of G is $|G| = m$.

We may allow in SLPs more general productions of the form $X_i \to w$ with $w \in (\Sigma \cup \{X_1, \ldots, X_{i-1}\})^*$. We may even allow exponential expressions of the form X_j^k for $j < i$ and a binary coded integer $k \in \mathbb{N}$ in the right-hand side w. Such a production can be replaced by $O(\log(k))$ many ordinary productions.

3 Querying the i-th Symbol

In this section, we study the following computational problem **Compressed Querying**:
INPUT: SLP G (over the terminal alphabet Σ), position $i \in \mathbb{N}$, and $a \in \Sigma$
QUESTION: $\mathrm{eval}(G)[i] = a$?
We prove that **Compressed Querying** is P-complete. This means that unless $\mathsf{P} = \mathsf{NC}$, where NC is the class of all problems that can be solved in polylogarithmic time using polynomially many processors, there does not exist an efficient parallel algorithm for **Compressed Querying**, see [9] for background on P-completeness. All reductions in this section are NC-reductions, i.e., they can be computed in polylogarithmic time with only polynomially many processors.

Theorem 1. Compressed Querying *is* P*-complete. Hardness for* P *even holds for a binary terminal alphabet.*

Proof. Membership in P is easy to see: first compute for every non-terminal X of the input SLP the length ℓ_X of the generated string $\mathrm{eval}(X)$. Now, if we have a production $X \to YZ$ and we want to determine $\mathrm{eval}(X)[i]$ then we first check whether $i \leq \ell_Y$. In this case we have to find $\mathrm{eval}(Y)[i]$. On the other hand, if $i > \ell_Y$, then we have to determine $\mathrm{eval}(Z)[i - \ell_X]$. This simple idea leads to a polynomial time algorithm.

We prove P-hardness by an NC-reduction from the P-complete problem **Super Increasing Subset Sum** [11]:

INPUT: Integers $w_1, \ldots, w_n, t$ in binary form such that $w_i > \sum_{j=1}^{i-1} w_j$ for all $1 \leq i \leq n$ (in particular $w_1 > 0$).

QUESTION: Do there exist $x_1, \ldots, x_n \in \{0,1\}$ such that $\sum_{i=1}^{n} x_i \cdot w_i = t$?

Thus, let $w_1, \ldots, w_n, t$ be integers such that $w_i > \sum_{j=1}^{i-1} w_j$. Let $g_1, \ldots, g_n \in \{0,1\}^*$ be defined as follows, where $s_j = w_1 + \cdots + w_j$ for $1 \leq j \leq n$:

$$g_1 = 10^{w_1 - 1}1 \qquad g_j = g_{j-1}0^{w_j - s_{j-1} - 1}g_{j-1} \text{ for } 2 \leq j \leq n$$

It is straightforward to construct from the instance $(w_1, \ldots, w_n, t)$ in NC an SLP that generates the string g_n. Note that $w_j > s_{j-1}$ and hence $w_j - s_{j-1} - 1 \geq 0$. Moreover, we claim that $|g_j| = s_j + 1$. This is certainly true for $j = 1$ since $s_1 = w_1$. For $j \geq 2$ we obtain inductively $|g_j| = 2|g_{j-1}| + w_j - s_{j-1} - 1 = 2s_{j-1} + 2 + w_j - s_{j-1} - 1 = s_j + 1$.

We claim that $g_n[t+1] = 1$ if and only if there exist $x_1, \ldots, x_n \in \{0,1\}$ such that $\sum_{i=1}^{n} x_i \cdot w_i = t$, which proves the theorem. For this, we prove by induction on j that for every $p \geq 0$: $g_j[p+1] = 1$ if and only if $\exists x_1, \ldots, x_j \in \{0,1\} : \sum_{i=1}^{j} x_i \cdot w_i = p$. If $j = 1$, then $g_1[p+1] = (10^{w_1-1}1)[p+1] = 1$ if and only if $p = 0$ or $p = w_1$, which proves the induction base. Now assume that $j \geq 2$. Then $g_j[p+1] = 1$ if and only if $(g_{j-1}0^{w_j - s_{j-1} - 1}g_{j-1})[p+1] = 1$ if and only if $(g_{j-1}[p+1] = 1$ or $g_{j-1}[p + 1 - |g_{j-1}| - w_j + s_{j-1} + 1] = 1)$ if and only if $(g_{j-1}[p+1] = 1$ or $g_{j-1}[p+1-s_{j-1}-1-w_j+s_{j-1}+1] = g_{j-1}[p+1-w_j] = 1)$ (since $|g_{j-1}| = s_{j-1}+1$). By induction, this is true if and only if

$$\exists x_1, \ldots, x_{j-1} \in \{0,1\} \left\{ \sum_{i=1}^{j-1} x_i \cdot w_i = p \quad \text{or} \quad \sum_{i=1}^{j-1} x_i \cdot w_i = p - w_j \right\}.$$

But this is equivalent to $\exists x_1, \ldots, x_j \in \{0,1\} : \sum_{i=1}^{j} x_i \cdot w_i = p$. $\square$

Note that in Thm. 1, P-hardness already holds for a binary alphabet. If we allow the terminal alphabet to be part of the input, then we can prove P-hardness even for a restricted form of SLPs, so called *restricted Lempel-Ziv encodings*, briefly RLZ-encodings [7]. For a given string $w \in \Sigma^+$, the *RLZ-factorization* of w is the unique factorization $w = f_1 f_2 \cdots f_n$ such that for every $i \geq 1$, f_i is either the longest non-empty prefix of $f_i f_{i+1} \cdots f_n$ such that there exists $1 \leq j \leq k < i$ with $f_i = f_j \cdots f_k$, or (if such a prefix does not exist) f_i is the first symbol of $f_i f_{i+1} \cdots f_n$. In this situation, the *RLZ-encoding* of w, briefly RLZ(w) is the sequence $c_1 c_2 \cdots c_n$, where $c_i = f_i$ if $f_i \in \Sigma$ or $c_i = [j,k]$ if $f_i \notin \Sigma$ and $f_i = f_j \cdots f_k$. Note that from RLZ(w) one can easily construct an SLP generating w.

Example 1. Let $w = abaababaabaababaababa$. Then the RLZ-factorization of w is $a \,|\, b \,|\, a \,|\, aba \,|\, baaba \,|\, ababaaba \,|\, ba$ and RLZ$(w) = aba[1,3][2,4][4,5][2,3]$.

The next theorem solves an open problem from [7], where a corresponding result for LZ-encoded input strings (see [7] for the definition) was shown. It should be noted that there are polynomial time transformations between RLZ- and LZ-encodings [7, Lemma 1], but by the results from [7] there cannot be an NC-transformation from LZ-encodings to RLZ-encodings unless $\mathsf{P} = \mathsf{NC}$.

Theorem 2. *The following problem is* P*-complete:*
INPUT: An alphabet Σ, a string $w \in \Sigma^*$ given by its RLZ-encoding, a position $i \in \mathbb{N}$, and $a \in \Sigma$
QUESTION: $w[i] = a$?

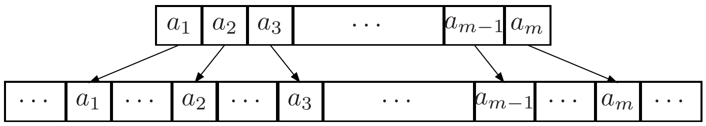
Proof. Membership in P follows from Thm. 1. For P-hardness we use almost the same construction as in the proof of Thm. 1. For a given instance $(w_1, \dots, w_n, t)$ of **Super Increasing Subset Sum** we define strings $g_1, \dots, g_n \in \{1, \$_1, \dots, \$_n\}^*$ as follows, where $s_j = w_1 + \dots + w_j$ for $1 \le j \le n$:

$$g_1 = 1\$_1^{w_1-1}1 \qquad g_j = g_{j-1}\$_j^{w_j-s_{j-1}-1}g_{j-1} \text{ for } 2 \le j \le n$$

The proof of Thm. 1 shows that $g_n[t+1] = 1$ if and only if there exist $x_1, \dots, x_n \in \{0,1\}$ such that $\sum_{i=1}^n x_i \cdot w_i = t$. It remains to prove that RLZ(g_n) can be constructed in NC from $(w_1, \dots, w_n, t)$. In the following let $\ell(i)$ for $i \in \mathbb{N}$ be the number of factors in the RLZ-factorization of a^i. One can show that $\ell(i) \in O(\log(i))$ and RLZ(a^i) can be calculated in NC from the binary encoding of i. Now we determine the number λ_i of factors of the RLZ-factorization of the string g_i. We have $\lambda_1 = 2 + \ell(w_1 - 1)$ and $\lambda_i = \lambda_{i-1} + \ell(w_i - s_{i-1} - 1) + 1$ for $i > 1$. Thus, $\lambda_i = (i+1) + \sum_{k=1}^i \ell(w_k - s_{k-1} - 1)$. Also the numbers λ_i $(1 \le i \le n)$ can be calculated in NC using the prefix sum algorithm. Now we can set in parallel for all $1 \le i \le n$ the factor from position $\lambda_{i-1} + 1$ to λ_i of RLZ(g_n) (where $\lambda_0 = 0$) to RLZ$(\$_i^{w_i-s_{i-1}-1})^{+\lambda_{i-1}}[1, \lambda_{i-1}]$, where RLZ$(w)^{+j}$ is the same as RLZ(w) but where j is added to all numbers. □

4 Complexity of Embedding

A string $p = a_1 \cdots a_m$ can be *embedded* into a string $t = b_1 \cdots b_n$ $(a_i, b_j \in \Sigma)$, briefly $p \hookrightarrow t$, if there exist positions $1 \le i_1 < i_2 < \dots < i_m \le n$ such that $b_{i_k} = a_k$ for $1 \le k \le m$. We also say that p is a *subsequence* of t, see the following diagram:


a_1
a_2
a_3
$\cdots$
a_{m-1}
a_m
$\cdots$ a_1 $\cdots$ a_2 $\cdots$ a_3 $\cdots$ a_{m-1} $\cdots$ a_m $\cdots$

In this section, we study the complexity of the following problem **Fully Compressed Embedding**, for short **Embedding**:
INPUT: SLPs P and T
QUESTION: eval$(P) \hookrightarrow$ eval(T)?
The following upper bound for **Embedding** is easy to prove:

Proposition 1. **Embedding** *belongs to* PSPACE.

Proof. The straightforward greedy algorithm that solves the embedding problem for uncompressed strings in linear time results in a PSPACE-algorithm for SLP-compressed strings. The crucial observation is that a position in a string, which is represented by an SLP, can be stored in polynomial space with respect to the size of the SLP. □

A simple greedy algorithm for checking $\mathrm{eval}(P) \hookrightarrow \mathrm{eval}(T)$ can be easily implemented within the time bound $|\mathrm{eval}(P)| \cdot |T|^{O(1)} \leq 2^{O(|P|)} \cdot |T|^{O(1)}$. This shows in particular that **Embedding** is fixed parameter tractable in the sense of [5], when the size of the pattern-SLP is chosen as the parameter (which is reasonable, because in most pattern matching applications the pattern is much smaller than the text).

Our main result states that **Embedding** is hard for the complexity class Θ_2^p. In Sec. 4.1, we prove NP-hardness. Then, in Sec. 4.2 we show how to simulate boolean operations with **Embedding**. From this, we deduce hardness for Θ_2^p in Sec. 4.3.

4.1 NP-Hardness of Embedding

Let us recall the well-known NP-complete problem **Subset Sum** (see [6]):
INPUT: Integers $w_1, \ldots, w_n, t$ in binary form
QUESTION: Do there exist $x_1, \ldots, x_n \in \{0, 1\}$ with $\sum_{i=1}^n x_i \cdot w_i = t$?

Theorem 3. Embedding *is* NP-*hard.*

Proof. We prove the theorem by a polynomial time reduction from **Subset Sum** to **Embedding**. Let $t, \overline{w} = (w_1, \ldots, w_n)$ be input data for **Subset Sum**. W.l.o.g. assume that $n \geq 2$. We are going to construct SLPs G and H such that there exists a subset of $\{w_1, \ldots, w_n\}$ with sum equal to t if and only if $\mathrm{eval}(G) \hookrightarrow \mathrm{eval}(H)$.

We begin with some notation. Let $s = w_1 + \cdots + w_n$ and $N = 2^n s$. We can assume that $t < s$. Let $x \in \{0, 1, \ldots, 2^n - 1\}$ be an integer. With x_i $(1 \leq i \leq n)$ we denote the i-th bit in the binary representation of x, where x_1 is the least significant bit. Thus, $x = \sum_{i=1}^n x_i 2^{i-1}$. We define $x \circ \overline{w} = \sum_{i=1}^n x_i w_i$. Hence, $x \circ \overline{w}$ is the sum of the subset of $\{w_1, \ldots, w_n\}$ encoded by the integer x. Hence, $t, \overline{w}$ is a positive instance of **Subset Sum** if and only if $\exists x \in \{0, \ldots, 2^n - 1\} : x \circ \overline{w} = t$. We now define strings g and h as follows:

$$h_1 = \prod_{x=0}^{2^n-1} (10^s) = (10^s)^{2^n} \qquad h_2 = 0^{2N} \qquad h_3 = \prod_{x=0}^{2^n-1} (0^{x \circ \overline{w}} 1 0^{s - x \circ \overline{w}})$$

$$h_4 = 0^{t+1} \qquad h_0 = h_1 h_2 h_3 h_4 \qquad h = h_0^{5N}$$

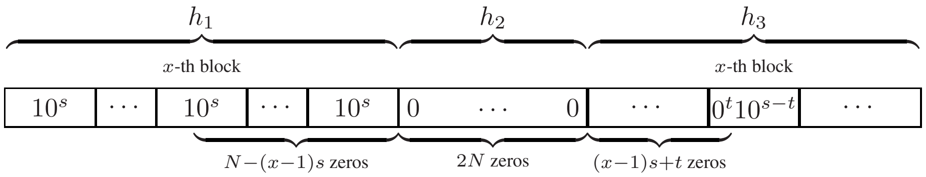
$$g_0 = 10^{3N+t} 10^{N+1} \qquad g = g_0^{5N-1}$$

We use the symbol $\prod$ to denote the concatenation of the corresponding words performed in the order $x = 0, \ldots, 2^n - 1$.

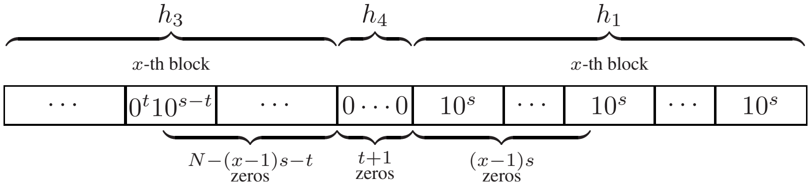
We first claim that the strings g and h can be generated by SLPs of polynomial size with respect to the size of the input $t, \overline{w}$. Note that with only one exception, namely the definition of h_3, only a constant number of concatenations and integer exponents with polynomially many bits are used in the definition of g and h. These constructions can be directly realized by SLPs. Finally, a construction of a polynomial size SLP for h_3 was presented in [12].

Now we prove that $g \hookrightarrow h$ if and only if $\exists x \in \{0, \ldots, 2^n - 1\} : x \circ \overline{w} = t$. First assume that there is $x \in \{0, \ldots, 2^n - 1\}$ such that $x \circ \overline{w} = t$. Consider the prefix $h_1 h_2 h_3 h_4 h_1$ of h. We can embed $g_0 = 10^{3N+t} 10^{N+1}$ into $h_1 h_2 h_3 h_4 h_1$: map the initial

1 of g_0 to the x-th block 10^s of h_1. Since $x \circ \overline{w} = t$, the number of 0's in $h_1h_2h_3$ between the 1 in the x-th block 10^s of h_1 and the x-th block $0^{x\circ\overline{w}}10^{s-x\circ\overline{w}} = 0^t10^{s-t}$ of h_3 is precisely $N-(x-1)s+2N+(x-1)s+t = 3N+t$, see the following diagram:

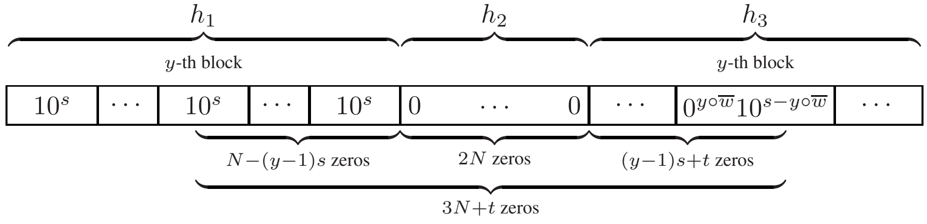


To these $3N+t$ many 0's we map the first $3N+t$ many 0's of g_0. Then the second 1 of g_0 is mapped to the 1 in the x-th block 0^t10^{s-t} of h_3. The next $N+1$ many 0's following this 1 are used for embedding the remaining $N+1$ many 0's of g_0. The crucial point is that after this embedding, we again arrive at the 1 in the x-th block 10^s of h_1, see the following diagram:

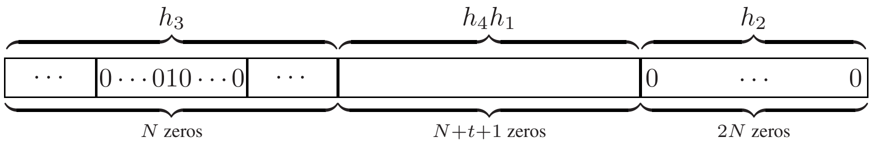


This observation shows that g_0^k can be embedded into $h_0^{k+1} = (h_1h_2h_3h_4)^{k+1}$ for every $k \geq 1$. In particular $g = g_0^{5N-1} \hookrightarrow h_0^{5N} = h$.

Next, we prove the reverse direction. Assume that $g \hookrightarrow h$. We have to show that there is $x \in \{0, \ldots, 2^n-1\}$ such that $x \circ \overline{w} = t$. In order to deduce a contradiction, assume that $x \circ \overline{w} \neq t$ for all $x \in \{0, \ldots, 2^n-1\}$. It turns out that not every 0 in h can be the image of a 0 from g under our embedding $g \hookrightarrow h$. Let us estimate the total number of such unused 0's. Our embedding $g \hookrightarrow h$ consists of $5N-1$ disjoint embeddings of g_0 into h. There are two 1's in g_0 and there are exactly $3N+t$ many 0's between them. We claim that there is no pair of two 1's with exactly $3N+t$ many 0's between them in h. In order to prove this, we consider two 1's in h and make a case distinction on the position of the first 1. First assume that the left 1 belongs to h_1. More precisely, assume that the left 1 is the 1 in the y-th block 10^s of h_1. By reading exactly $3N+t$ many 0's in h, we arrive at position $t+1$ (if $t < y \circ \overline{w}$) or $t+2$ (if $t > y \circ \overline{w}$) in the y-th block of h_3; note that $t < s$. But since $y \circ \overline{w} \neq t$, this position cannot contain a 1. This proves the case that the left 1 belongs to h_1. The following diagram visualizes the situation (where we assume that $t > y \circ \overline{w}$):



In the second case, the left 1 in our pair is situated in h_3. Then, by reading exactly $3N+t$ many 0's in h, we end up in h_2, which does not contain 1's at all:



We have now shown that for each embedding of g_0 in h between the images of the two 1's in g_0, there must be at least $3N + t + 1$ many 0's in h. Thus, for every embedding of $g_0 = 10^{3N+t}10^{N+1}$ in h we need at least $3N + t + 1 + N + 1 = 4N + t + 2$ many 0's in h. Since $g = g_0^{5N-1}$, we need at least

$$(4N + t + 2) \cdot (5N - 1) = 5N \cdot (4N + t + 1) + (N - t - 2) > 5N \cdot (4N + t + 1)$$

many 0's in h. For the last inequality note that $N = s \cdot 2^n \geq 4s > s + 2 > t + 2$. We obtain a contradiction, because from the construction of h, we see that h contains precisely $5N \cdot (4N + t + 1)$ many 0's. □

4.2 Simulating Boolean Operations

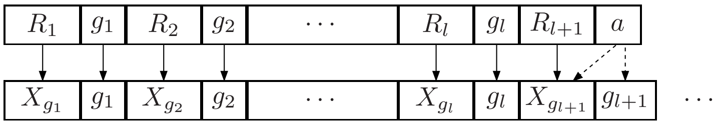
Proposition 2. *For SLPs G and H over a terminal alphabet Σ, $|\Sigma| \geq 1$, we can construct in polynomial time SLPs G' and H' over the terminal alphabet Σ such that*

$$\mathrm{eval}(G) \hookrightarrow \mathrm{eval}(H) \quad \Leftrightarrow \quad \mathrm{eval}(G') \not\hookrightarrow \mathrm{eval}(H'). \tag{1}$$

Proof. Let $\mathrm{eval}(G) = g_1 \cdots g_k$ and $\mathrm{eval}(H) = h_1 \cdots h_m$. For $a \in \Sigma$ let $X_a = (a_1 \cdots a_n)^{m+1}$, where $\{a_1, \ldots, a_n\} = \Sigma \setminus \{a\}$ (the order on $\Sigma \setminus \{a\}$ is arbitrary here; if $n = 0$, then $X_a = \varepsilon$). Let $a \in \Sigma$ be arbitrary and let G' and H' be SLPs with

$$\mathrm{eval}(G') = \mathrm{eval}(H)a = h_1 \cdots h_m a \quad \text{and} \quad \mathrm{eval}(H') = X_{g_1} g_1 \cdots X_{g_k} g_k.$$

These SLPs can be constructed in polynomial time from G and H. For G' this is clear. For H' we have to replace every terminal symbol a in G by a new nonterminal A and add the rule $A \to X_a a$. It remains to show (1). First assume that $\mathrm{eval}(G) \not\hookrightarrow \mathrm{eval}(H)$. Then we can write $\mathrm{eval}(H) = R_1 g_1 \cdots R_l g_l R_{l+1}$, where $l < k$ and for $1 \leq i \leq l + 1$, the word R_i does not contain the letter g_i. Since $|R_i| \leq m$, for every $1 \leq i \leq l + 1$ we have $R_i \hookrightarrow X_{g_i}$. Thus, we can embed the prefix $\mathrm{eval}(H) = R_1 g_1 \cdots R_l g_l R_{l+1}$ of $\mathrm{eval}(G')$ into the prefix $X_{g_1} g_1 \cdots X_{g_l} g_l X_{g_{l+1}}$ of $\mathrm{eval}(H')$. The final letter a of $\mathrm{eval}(G')$ can be either also mapped to $X_{g_{l+1}}$ (if $a \neq g_{l+1}$; here it is important that $|X_{g_{l+1}}| > m$ so that R_{l+1} does not completely occupy $X_{g_{l+1}}$) or it can be mapped to g_{l+1} (if $a = g_{l+1}$):



Now assume that $\mathrm{eval}(G) \hookrightarrow \mathrm{eval}(H)$. Then we can write $\mathrm{eval}(H) = R_1 g_1 \cdots R_k g_k R$, where for $1 \leq i \leq k$, the word R_i does not contain the letter g_i. We claim that

$$\forall 1 \leq i \leq k : R_1 g_1 \cdots R_i g_i \not\hookrightarrow X_{g_1} g_1 \cdots X_{g_{i-1}} g_{i-1} X_{g_i}. \tag{2}$$

Our proof goes by induction on i. In the case $i = 1$ this follows, since g_1 does not occur in X_{g_1}. For the induction step assume that (2) is true for some $i \geq 1$ and that moreover

$$R_1 g_1 \cdots R_i g_i R_{i+1} g_{i+1} \hookrightarrow X_{g_1} g_1 \cdots X_{g_{i-1}} g_{i-1} X_{g_i} g_i X_{g_{i+1}}. \tag{3}$$

Recall that the last symbol g_{i+1} of $R_1 g_1 \cdots R_{i+1} g_{i+1}$ does not occur in the suffix $X_{g_{i+1}}$ of $X_{g_1} g_1 \cdots X_{g_i} g_i X_{g_{i+1}}$. Thus, (3) implies that already $R_1 g_1 \cdots R_i g_i R_{i+1} g_{i+1} \hookrightarrow X_{g_1} g_1 \cdots X_{g_{i-1}} g_{i-1} X_{g_i} g_i$ and hence $R_1 g_1 \cdots R_i g_i R_{i+1} \hookrightarrow X_{g_1} g_1 \cdots X_{g_{i-1}} g_{i-1} X_{g_i}$. But this contradicts (2).

For $i = k$, (2) implies $R_1 g_1 \cdots R_k g_k \not\hookrightarrow X_{g_1} g_1 \cdots X_{g_{k-1}} g_{k-1} X_{g_k}$. But then $\text{eval}(G') = R_1 g_1 \cdots R_k g_k Ra \not\hookrightarrow X_{g_1} g_1 \cdots X_{g_{k-1}} g_{k-1} X_{g_k} g_k = \text{eval}(H')$. □

Thm. 3 and Prop. 2 immediately imply that **Embedding** is also coNP-hard.

Proposition 3. *For SLPs G_1, H_1, G_2, H_2 over a terminal alphabet Σ, $|\Sigma| \geq 2$, we can construct in polynomial time SLPs G, H over the terminal alphabet Σ such that*

$$(\text{eval}(G_1) \hookrightarrow \text{eval}(H_1) \text{ and } \text{eval}(G_2) \hookrightarrow \text{eval}(H_2)) \quad \Leftrightarrow \quad \text{eval}(G) \hookrightarrow \text{eval}(H).$$

Proof. W.l.o.g. assume that G_1 and G_2 (resp. H_1 and H_2) have disjoint sets of non-terminals. Let S_i (resp. T_i) be the start non-terminal of G_i (resp. H_i). Let $N = 1 + \max\{|\text{eval}(H_1)|, |\text{eval}(H_2)|\}$. Then G (resp. H) contains all productions of G_1 and G_2 (resp. H_1 and H_2) and the additional production $S \to S_1 1^N 0 1^N S_2$ (resp. $T \to T_1 1^N 0 1^N T_2$), where $0, 1 \in \Sigma$. Here, S (resp. T) is the start non-terminal of G (resp. H). Thus,

$$\begin{aligned} \text{eval}(G) &= \text{eval}(G_1)\, 1^N\, 0\, 1^N\, \text{eval}(G_2) \text{ and} \\ \text{eval}(H) &= \text{eval}(H_1)\, 1^N\, 0\, 1^N\, \text{eval}(H_2). \end{aligned}$$

Clearly, if $\text{eval}(G_1) \hookrightarrow \text{eval}(H_1)$ and $\text{eval}(G_2) \hookrightarrow \text{eval}(H_2)$, then we have $\text{eval}(G) \hookrightarrow \text{eval}(H)$. For the other direction note that if $\text{eval}(G_1) 1^N 0 1^N \text{eval}(G_2)$ can be embedded into $\text{eval}(H_1) 1^N 0 1^N \text{eval}(H_2)$, then by the choice of N, the 0 at position $|\text{eval}(G_1)| + N + 1$ in $\text{eval}(G_1) 1^N 0 1^N \text{eval}(G_2)$ can neither be mapped to the prefix $\text{eval}(H_1)$ nor to the suffix $\text{eval}(H_2)$ of $\text{eval}(H)$. Thus, this 0 has to be mapped to the 0 at position $|\text{eval}(H_1)| + N + 1$ in $\text{eval}(H_1) 1^N 0 1^N \text{eval}(H_2)$. This implies that both $\text{eval}(G_1) \hookrightarrow \text{eval}(H_1)$ and $\text{eval}(G_2) \hookrightarrow \text{eval}(H_2)$. □

Proposition 4. *For SLPs G_1, H_1, G_2, H_2 over a terminal alphabet Σ, $|\Sigma| \geq 2$, we can construct in polynomial time SLPs G, H over the terminal alphabet Σ such that*

$$(\text{eval}(G_1) \hookrightarrow \text{eval}(H_1) \text{ or } \text{eval}(G_2) \hookrightarrow \text{eval}(H_2)) \quad \Leftrightarrow \quad \text{eval}(G) \hookrightarrow \text{eval}(H).$$

Proof. W.l.o.g. assume that G_1, G_2, H_1, and H_2 have pairwise disjoint sets of non-terminals. Let S_i (resp. T_i) be the start non-terminal of G_i (resp. H_i). Let $N = 1 + |\text{eval}(G_1)| + |\text{eval}(G_2)|$. Then G contains all productions of G_1 and G_2 and the additional production $S \to S_1 0 1^N 0 S_2$. The SLP H contains all productions of G_1, H_1, G_2, H_2 and the additional production $T \to T_1 0 1^N S_1 0 S_2 1^N 0 T_2$. Thus, we have

$$\begin{aligned} \text{eval}(G) &= \text{eval}(G_1)\, 0\, 1^N\, 0\, \text{eval}(G_2) \text{ and} \\ \text{eval}(H) &= \text{eval}(H_1)\, 0\, 1^N\, \text{eval}(G_1)\, 0\, \text{eval}(G_2)\, 1^N\, 0\, \text{eval}(H_2). \end{aligned}$$

Clearly, if $\mathrm{eval}(G_1) \hookrightarrow \mathrm{eval}(H_1)$ or $\mathrm{eval}(G_2) \hookrightarrow \mathrm{eval}(H_2)$, then $\mathrm{eval}(G) \hookrightarrow \mathrm{eval}(H)$. For the other direction assume that $\mathrm{eval}(G) = \mathrm{eval}(G_1)\, 0\, 1^N\, 0\, \mathrm{eval}(G_2)$ can be embedded into $\mathrm{eval}(H) = \mathrm{eval}(H_1)\, 0\, 1^N\, \mathrm{eval}(G_1)\, 0\, \mathrm{eval}(G_2)\, 1^N\, 0\, \mathrm{eval}(H_2)$. Consider the 1^N-block of $\mathrm{eval}(G)$. If a 1 from this block is mapped to the prefix $\mathrm{eval}(H_1)$ of $\mathrm{eval}(H)$, then $\mathrm{eval}(G_1) \hookrightarrow \mathrm{eval}(H_1)$. If a 1 from the 1^N-block of $\mathrm{eval}(G)$ is mapped to the first 1^N-block of $\mathrm{eval}(H)$, then the 0 at position $|\mathrm{eval}(G_1)|+1$ in $\mathrm{eval}(G)$ cannot be mapped to the right of the 0 at position $|\mathrm{eval}(H_1)|+1$ in $\mathrm{eval}(H)$. But then again the prefix $\mathrm{eval}(G_1)$ of $\mathrm{eval}(G)$ is embedded into the prefix $\mathrm{eval}(H_1)$ of $\mathrm{eval}(H)$. Completely analogously it follows that if a 1 from the 1^N-block of $\mathrm{eval}(G)$ is mapped to the suffix $\mathrm{eval}(H_2)$ of $\mathrm{eval}(H)$ or to the second 1^N-block of $\mathrm{eval}(H)$, then $\mathrm{eval}(G_2) \hookrightarrow \mathrm{eval}(H_2)$. The only remaining case, namely that every 1 in the 1^N-block of $\mathrm{eval}(G)$ is mapped into $\mathrm{eval}(G_1)\, 0\, \mathrm{eval}(G_2)$ cannot occur, since $N > |\mathrm{eval}(G_1)\mathrm{eval}(G_2)|$. □

4.3 Hardness for Θ_2^p

Recall that Θ_2^p is the class of all problems that can be accepted by a deterministic polynomial time machine with access to an oracle from NP and such that furthermore all questions to the oracle are asked in parallel [23].

Proposition 5. *If $A \subseteq \{0,1\}^*$ is* NP*-complete, then the following problem is Θ_2^p-complete:*

INPUT: A boolean circuit C (i.e., a circuit with AND-gates, OR-gates, NOT-gates, and input gates), where every input gate g is labeled with a word $w(g) \in \{0,1\}^*$.

QUESTION: Does C evaluate to true when every input gate g evaluates to true (resp. false) if $w(g) \in A$ (resp. $w(g) \notin A$)?

Proof. For membership in Θ_2^p note that we can evaluate all input gates of C in parallel by using the language A as an oracle. Then, the whole circuit can be evaluated in polynomial time. Hardness for Θ_2^p follows from a result from [23]: It is Θ_2^p-complete to decide for a given list of strings $w_1, w_2, \ldots, w_n \in \{0,1\}^*$, whether the number $|\{i \mid w_i \in A\}|$ is odd. By taking a boolean circuit for parity, this problem can be easily encoded into a boolean circuit with A-instances at input gates. □

Theorem 4. *Even for SLPs over a binary terminal alphabet,* **Embedding** *is Θ_2^p-hard.*

Proof. Let C be a circuit with input gates labeled with instances of the NP-complete **Subset Sum** problem. By the usual doubling argument, we can assume that negation gates only occur directly above input gates. We first define inductively for every gate c strings $u(c)$ and $v(c)$ and then argue that (i) c evaluates to true if and only if $u(c) \hookrightarrow v(c)$ and (ii) $u(c)$ and $v(c)$ can be generated by "small" SLPs. If c is an unnegated input gate that is labeled with the **Subset Sum** instance I then $u(c) = g$ and $v(c) = h$, where g and h are the two strings that are constructed from I in the proof of Thm. 3. If c is a negated input gate that is labeled with the **Subset Sum** instance I, then again we first construct from I the words g and h as described in the proof of Thm. 3. Then we apply the construction from the proof of Prop. 2 to g and h and assign the resulting strings to $u(c)$ and $v(c)$, respectively. For AND- and OR-gates we use the constructions from the proofs of Prop. 3 and 4, resp.: If c is an AND-gate with inputs c_1 and c_2, then

$$u(c) = u(c_1)\,1^N\,0\,1^N\,u(c_2) \text{ and } v(c) = v(c_1)\,1^N\,0\,1^N\,v(c_2), \tag{4}$$

where $N = 1 + \max\{|v(c_1)|, |v(c_2)|\}$. If c is an OR-gate with inputs c_1 and c_2, then

$$u(c) = u(c_1)\,0\,1^N\,0\,u(c_2) \text{ and } v(c) = v(c_1)\,0\,1^N\,u(c_1)\,0\,u(c_2)\,1^N\,0\,v(c_2), \tag{5}$$

where $N = 1 + |u(c_1)| + |u(c_2)|$. From Thm. 3 and Prop. 2–4 it follows immediately that C evaluates to true if and only if $u(o) \hookrightarrow v(o)$, where o is the output gate of C.

It remains to argue that for every gate c, the strings $u(c)$ and $v(c)$ can be generated by SLPs of size polynomially bounded in the size of the circuit C (which is the number of gates plus the size of all **Subset Sum** instances at the input gates of C). Note that if we define $n(c) = \max\{|u(c)|, |v(c)|\}$ then we have $n(c) \leq 8 \cdot \max\{n(c_1), n(c_2)\} + 5$ in case c is an AND- or OR-gate with inputs c_1 and c_2.[1] It follows that $n(c)$ is bounded exponentially in the size of the circuit C. Moreover, we can calculate the binary representations of the lengths $|u(c)|$ and $|v(c)|$ for every gate c in polynomial time. Thus, we can construct SLPs of polynomial size for the factors 1^N in (4) and (5). This implies that for every gate c, $u(c)$ and $v(c)$ can be generated by SLPs of polynomial size. □

Let us close this paper with a corollary of Thm. 4. In the problem **Longest Common Subsequence** (**LCS**) (resp. **Shortest Common Supersequence** (**SCS**)), one asks for a finite set R of strings and $n \in \mathbb{N}$ whether there is a string w with $|w| \geq n$ and $\forall v \in R : w \hookrightarrow v$ (resp. $|w| \leq n$ and $\forall v \in R : v \hookrightarrow w$). These problems are known to be NP-complete, but for $|R| = 2$ they can be solved in polynomial time (see [6]). For SLP-encoded input strings, **LCS** and **SCS** can be both solved in PSPACE.

Corollary 1. *The problems* **LCS** *and* **SCS** *for SLP-encoded input strings are* Θ_2^p*-hard, even if* $|R| = 2$ *for the input set* R.

Proof. For $u, v \in \Sigma^*$ we have $u \hookrightarrow v$ if and only if $(\{u, v\}, |u|)$ (resp. $(\{u, v\}, |v|)$) is a true instance of **LCS** (resp. **SCS**). Hence, the corollary follows from Thm. 4. □

5 Open Problems

The main open problem that remains from this paper concerns the precise complexity of **Embedding**. Our results leave a gap from Θ_2^p to PSPACE. In Thm. 2 (P-completeness of querying RLZ-encoded input strings) it is open, whether the underlying alphabet can be fixed to, e.g., a binary alphabet.

Acknowledgments. This work was done during a visit of the first author at University of Stuttgart, Germany, which was supported by the DFG project GELO. The first author was also supported by grants from the projects INTAS 04-77-7173 and NSh-8464.2006.1.

[1] Such a bound would not hold for a NOT-gate, since one application of the construction for Prop. 2 may lead to a quadratic blow-up in the size of the generated strings. Therefore we have to assume that NOT-gates only appear at input gates.

References

1. A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *J. Comput. Syst. Sci*, 52(2):299–307, 1996.
2. M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: From word to circuit evaluation. *SIAM J. Comput.*, 26(1):138–152, 1997.
3. P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *J. Comput. Syst. Sci.*, 65(2):332–350, 2002.
4. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005.
5. R. G. Downey and M. R. Fellows. *Parametrized Complexity*. Springer, 1999.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman, 1979.
7. L. Gasieniec, A. Gibbons, and W. Rytter. Efficiency of fast parallel pattern searching in highly compressed texts. In *Proc. MFCS'99*, LNCS 1672, pages 48–58. Springer, 1999.
8. L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding (extended abstract). In *Proc. SWAT 1996*, LNCS 1097, pages 392–403. Springer, 1996.
9. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford Univ. Press, 1995.
10. D. Gushfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge Univ. Press, 1999.
11. H. J. Karloff and W. L. Ruzzo. The iterated mod problem. *Inf. Comput.*, 80(3):193–204, 1989.
12. M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210 – 1240, 2006.
13. N. Markey and P. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inf. Process. Lett.*, 90(1):3–6, 2004.
14. M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. CPM 97*, LNCS 1264, pages 1–11. Springer, 1997.
15. G. Navarro. Regular expression searching on compressed text. *J. Discrete Algorithms*, 1(5–6):423–443, 2003.
16. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
17. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, LNCS 855, pages 460–470. Springer, 1994.
18. W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 262–272. Springer, 1999.
19. W. Rytter. Algorithms on compressed strings and arrays. In *Proc. SOFSEM'99*, LNCS 1725, pages 48–65. Springer, 1999.
20. W. Rytter. Compressed and fully compressed pattern matching in one and two dimensions. *Proceedings of the IEEE*, 88(11):1769–1778, 2000.
21. W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1–3):211–222, 2003.
22. W. Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In *Proc. ICALP 2004*, LNCS 3142, pages 15–27. Springer, 2004.
23. K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theor. Comput. Sci.*, 51:53–80, 1987.
24. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.

Lempel-Ziv Dimension for Lempel-Ziv Compression[*]

Maria Lopez-Valdes

Departamento de Informática e Ing. de Sistemas, María de Luna 1,
Universidad de Zaragoza. 50018 Zaragoza, Spain
marlopez@unizar.es

Abstract. This paper describes the Lempel-Ziv dimension (Hausdorff like dimension inspired in the LZ78 parsing), its fundamental properties and relation with Hausdorff dimension. It is shown that in the case of individual infinite sequences, the Lempel-Ziv dimension matches with the asymptotical Lempel-Ziv compression ratio. This fact is used to describe results on Lempel-Ziv compression in terms of dimension of complexity classes and vice versa.

1 Introduction

Lutz [9] developed effective dimension (a Hausdorff like dimension) to quantitatively analyze the structure of complexity classes. Later, other authors have developed dimensions such as constructive or finite-state dimension and have found new connections with information theory. In particular, the motivation of this paper is the relation between dimension and compression. In this context, it was shown that polynomial-time and finite-state dimension can be characterized by the best compression ratio of polynomial-time and finite-state compressors respectively [8,4]. As consequence, some results on dimension of complexity classes can be interpreted as compressibility results.

This paper focusses on the relation between dimension and the Lempel-Ziv compressor (LZ78) [12,11]. This compression algorithm is probably the most widely studied of the universal compressors (compressors that do not depend on the distribution of the sequence source). Previous results have shown that polynomial-time and finite-state dimension are respectively a lower and upper bound of the asymptotical Lempel-Ziv compression ratio [8,4]. However, by defining a dimension that matches the Lempel-Ziv compression ratio, results on Lempel-Ziv compression could be directly used to determine the dimension of some complexity classes. Alternatively, researchers in the compression domain have pointed out that it would be interesting to define a Hausdorff like dimension inspired in the LZ78 parsing, and to see in which cases this dimension would match with the Hausdorff dimension. This is because it is expected

[*] This research was supported by Spanish Government MEC project TIN2005-08832-C03-02.

that dimension would allow to deal with open problems related with Lempel-Ziv compression.

This paper describes the Lempel-Ziv dimension (Hausdorff like dimension inspired in the LZ78 parsing), its fundamental properties and relation with Hausdorff dimension. This dimension is naturally included in the existing hierarchy of dimensions defined until now. Furthermore, in the case of individual sequences, the Lempel-Ziv dimension matches with the asymptotical Lempel-Ziv compression ratio. This result is used to show some applications on dimension of complexity classes. Finally, results on dimension and ergodic theory are used to partially solve the open question raised by Lutz and others [6,7] on the one-bit catastrophe in the LZ78 compressor.

This paper is distributed as follows. Section 2 outlines the preliminaries on the Lempel-Ziv compressor, measure, entropy and dimension. Section 3 describes the polynomial-time and finite-state dimension and some results related with compression. Section 4 develops the Lempel-Ziv dimension and fundamental properties. Section 5 describes results on Lempel-Ziv compression in terms of results on dimension of complexity classes and vice versa (in particular the one-bit catastrophe in the LZ78 compressor).

2 Preliminaries

Let a *string* be a finite and binary sequence $w \in \{0,1\}^*$. Let $|w|$ denote the length of a string and λ the empty string. The *Cantor space* $\mathbf{C}$ is the set of all infinite binary sequences. Let $x[i \dots j]$ for $0 \leq i \leq j$ denote the i-th through the j-th bits of x, where $x \in \{0,1\}^* \cup \mathbf{C}$. Let wx denote the concatenation of the string w and the string or sequence x. Let $w \sqsubseteq x$ denote that w is a *prefix* of x.

Let a *parsing* of a string $w \in \{0,1\}^*$ be a partition of w into *phrases* $w_1, w_2, \dots, w_n$ such that $w_1 w_2 \dots w_n = w$. Let a *distinct parsing* of a string $w \in \{0,1\}^*$ be a parsing of w such that no phrase, except possibly the last phrase, is the same as an earlier phrase. Let a *valid distinct parsing* of a string $w \in \{0,1\}^*$ be a distinct parsing of w such that if w_i is a phrase in the string w, then every prefix of w_i appears before w_i in the distinct parsing. Note that each string w has an unique valid distinct parsing.

The LZ78 *compression algorithm* encodes a given string with its valid distinct parsing. This is done by replacing each phrase with a code word representing a pointer and a bit. The pointer indicates the longest proper prefix of the phrase and the bit is the last bit of the phrase. Together, they completely specify the phrase being encoded. The output of the compression algorithm on the string w is denoted LZ(w). (See [11,12] for more details).

2.1 Probability Measures on C and Entropy

Let C_w be the *cylinder* generated by a string $w \in \{0,1\}^*$, given by $C_w = \{S \in \mathbf{C} \mid w \sqsubseteq S\}$. Let $\mathcal{F}$ be the σ-algebra generated from the cylinder sets of $\mathbf{C}$. Let $\nu : \mathcal{F} \to [0,1]$ be a countable additive, nonnegative measure with total mass 1.

Then, the triple $(\mathbf{C}, \mathcal{F}, \nu)$ is known as a *probability space* and ν is a *probability measure* on $\mathbf{C}$.

A measure ν is *stationary* if $\nu(T^{-1}X) = \nu(X)$ for all $X \in \mathcal{F}$, where T is the left-shift on $\mathbf{C}$ (i.e. for $b \in \{0,1\}$ and $S \in \mathbf{C}$, $T(bS) := S$). A measure ν is *ergodic* if any T-invariant set X (any set such that $T^{-1}X = X$) has $\nu(X) = 0$ or $\nu(X) = 1$.

On one hand, each probability measure ν is identified on $\mathbf{C}$ with a function $\mu : \{0,1\}^* \to [0,1]$ defined by $\mu(w) = \nu(C_w)$. This function μ verifies the following Kolmogorov's consistency conditions

$$(i)\ \ \mu(\lambda) = 1.$$
$$(ii)\ \mu(w0) + \mu(w1) = \mu(w) \ \text{ for all } w \in \{0,1\}^*.$$

On the other hand, by Kolmogorov's Existence Theorem [1], for each $\mu : \{0,1\}^* \to [0,1]$ satisfying the consistence conditions, there exists a unique probability measure ν on $\mathbf{C}$ such that $\nu(C_w) = \mu(w)$. Then, for simplicity, a function μ verifying (i) and (ii) will be referred also as a probability measure on $\mathbf{C}$.

Let the *entropy* of an stationary measure μ on $\mathbf{C}$ be:

$$\mathcal{H}(\mu) = \lim_n \tfrac{\mathcal{H}_n(\mu)}{n}, \text{ where } \mathcal{H}_n(\mu) = \textstyle\sum_{w \in \{0,1\}^n} \mu(w) \log \tfrac{1}{\mu(w)}. \tag{1}$$

Theorem 1. *[3] (Entropy-rate Theorem) Let μ be a stationary ergodic probability measure on $\mathbf{C}$. Then, it exists a constant $h \geq 0$, such that*

$$\lim_n -\frac{1}{n} \log \mu(S[0 \ldots n-1]) = h$$

almost surely. This constant h is called the entropy rate *of μ.*

Theorem 2. *[3] For a stationary measure μ, the entropy $\mathcal{H}(\mu)$ is always defined. If μ is a stationary ergodic measure, then the entropy rate h and the entropy $\mathcal{H}(\mu)$ are equal.*

2.2 Gales and Hausdorff Dimension

Definition 1. *Let $s \in [0, \infty)$.*

1. *An s-*supergale *is a function $d : \{0,1\}^* \to [0, \infty)$ satisfying*

$$d(w) \geq 2^{-s}[d(w0) + d(w1)]$$

   for all $w \in \{0,1\}^*$.
2. *An s-supergale d* succeeds *on a sequence $S \in \mathbf{C}$ if*

$$\limsup_n d(S[0 \ldots n]) = \infty.$$

3. *The* success set *of d is $S^\infty[d] = \{S \in \mathbf{C} \mid d \text{ succeeds on } S\}$.*

In 2003, Lutz [9] proved a characterization of classical Hausdorff dimension in terms of the s-supergales.

Theorem 3. *[9] For every $X \subseteq \mathbf{C}$,*

$$\dim_H(X) = \inf\{s \in [0,\infty) \mid \exists\ s\text{-supergale } d \text{ s.t. } X \subseteq S^\infty[d]\}.$$

3 Polynomial-Time Dimension, Finite-State Dimension and Lempel-Ziv Compression

This section describes the polynomial-time and finite-state dimension and some results related with compression.

Based in the characterization of classical Hausdorff dimension, Lutz developed *resource-bounded dimension* [9] by introducing a resource-bound Δ and requiring the supergales to be Δ-computable.

$$\dim_\Delta(X) = \inf\{s \in [0,\infty) \mid \exists\ \Delta\text{-computable } s\text{-supergale } d \text{ s.t. } X \subseteq S^\infty[d]\}.$$

This is the case of *polynomial-time dimension* that is defined by restricting attention to polynomial-time s-supergales (see [9] for further details).

Definition 2. *The* polynomial-time dimension *of $X \subseteq \mathbf{C}$ is*

$$\dim_{\mathrm{p}}(X) = \inf\{s \in [0,\infty) \mid \exists\ \mathrm{p}\text{-}computable\ s\text{-}supergale\ d\ s.t.\ X \subseteq S^\infty[d]\}.$$

The polynomial-time dimension of a sequence $S \in \mathbf{C}$ is $\dim_{\mathrm{p}}(S) = \dim_{\mathrm{p}}(\{S\})$.

Also following this scheme, *finite-state dimension* is defined by restricting attention to s-supergales that are specified by finite-state devices (see [4] for a complete introduction).

Definition 3. *The* finite-state dimension *of a set $X \subseteq \mathbf{C}$ is*

$$\dim_{\mathrm{FS}}(X) = \inf\{s \in [0,\infty) \mid \exists\ \ finite\text{-}state\ s\text{-}supergale\ d\ s.t.\ X \subseteq S^\infty[d]\}.$$

The finite-state dimension of a sequence $S \in \mathbf{C}$ is $\dim_{\mathrm{FS}}(S) = \dim_{\mathrm{FS}}(\{S\})$.

The next proposition states that to add a string in the beginning of a sequence does not change its polynomial-time neither the finite-state dimension.

Proposition 1. *For all $S \in \mathbf{C}$ and $w \in \{0,1\}^*$,*

1. $\dim_{\mathrm{p}}(wS) = \dim_{\mathrm{p}}(S)$.
2. $\dim_{\mathrm{FS}}(wS) = \dim_{\mathrm{FS}}(S)$.

The polynomial-time dimension is characterized as the best asymptotic compression-ratio attained by some special class of polynomial-time compressors. Lempel-Ziv compressor is in this class. Then,

Proposition 2. *[8] For every $S \in C$,*

$$\dim_{\mathrm{p}}(S) \leq \liminf_{n} \frac{|\mathrm{LZ}(S[0 \ldots n-1])|}{n}.$$

In [4] the authors obtained the following compressibility characterization of finite-state dimension of a sequence,

$$\dim_{\mathrm{FS}}(S) = \rho_{\mathrm{FS}}(S)$$

where $\rho_{\mathrm{FS}}(S)$ is the best compression ratio attainable for the infinite sequence S by any information lossless finite-state compressor.

On the other side, it's well known [12,11] that

$$\liminf_{n} \frac{|\mathrm{LZ}(S[0 \ldots n-1])|}{n} \leq \rho_{\mathrm{FS}}(S).$$

Therefore, we can obtain the relationship between finite-state dimension and Lempel-Ziv compresion.

Proposition 3. *For every $S \in \mathbf{C}$,*

$$\liminf_{n} \frac{|\mathrm{LZ}(S[0 \ldots n-1])|}{n} \leq \dim_{\mathrm{FS}}(S).$$

Thus, the polynomial-time dimension and the finite-state dimension are respectively a lower and upper bound of the Lempel-Ziv compression-ratio.

4 Lempel-Ziv Dimension

This section develops Lempel-Ziv dimension and its fundamental properties. Notice that the previous section is a good example of the close relationship between dimension and compression. The Lempel-Ziv dimension allows to see the Lempel-Ziv compression under the dimension point of view.

The definition of the Lempel-Ziv dimension is motivated by [10]. In this paper, the constructive dimension is defined by taking advantage of the existence of an universal constructive subprobability measure [13]. This allows us to center the definition in a single family of supergales $\{\tilde{d}^s\}_{s \geq 0}$,

$$\mathrm{cdim}(X) = \inf\{s \in [0, \infty) \mid X \subseteq S^{\infty}[\tilde{d}^s]\}.$$

The Lempel-Ziv algorithm (LZ78) is universal and asymptotically optimal for finite-state compressors [11,12]. Therefore, as in the case of constructive dimension, we can define the Lempel-Ziv dimension in terms of a single family of supergales, $\{d^s_{\mathrm{LZ}}\}_{s \geq 0}$.

Definition 4. *For each $s \in [0, \infty)$, let the* Lempel-Ziv s-supergale d^s_{LZ} *be:*

$$d^s_{\mathrm{LZ}}(\lambda) = 1$$

$$d^s_{\mathrm{LZ}}(w) = \begin{cases} \frac{2^{s|w|}}{n!} \frac{\#\{i \in \{1 \ldots n\} \mid u \sqsubseteq w_i\}}{n} & \text{if} \quad w = w_1 w_2 \ldots w_n u \\ \frac{2^{s|w|}}{n!} & \text{if} \quad w = w_1 w_2 \ldots w_n \end{cases}$$

where $w_1, \dots w_n$ are the distinct phrases in the valid distinct parsing of w and $u = w_i$ for some $i \in \{1 \dots n\}$.

Remark 1. For every polynomial-time computable real s, the Lempel-Ziv s-supergale is polynomial-time computable.

Proposition 4. *The Lempel-Ziv s-supergale is optimal for the class of finite-state s-supergales. That is, there exists $\alpha > 0$ such that for all $s \in [0, \infty)$ and all d finite-state s-supergale,*

$$d^s_{\mathrm{LZ}}(w) \geq \alpha d(w),$$

for every $w \in \{0,1\}^*$ (long enough).

Remark 2. For all $s, t \in [0, \infty)$ and $w \in \{0,1\}^*$,

$$d^s_{\mathrm{LZ}}(w) 2^{-s|w|} = d^t_{\mathrm{LZ}}(w) 2^{-t|w|}.$$

Definition 5. *The* Lempel-Ziv dimension *of $X \subseteq \mathbf{C}$ is,*

$$\dim_{\mathrm{LZ}}(X) = \inf\{s \in [0, \infty) \mid X \subseteq S^\infty[d^s_{\mathrm{LZ}}]\}.$$

The Lempel-Ziv dimension *of a sequence $S \in \mathbf{C}$ is $\dim_{\mathrm{LZ}}(S) = \dim_{\mathrm{LZ}}(\{S\})$.*

Remark 3. For all $X \subseteq Y \subseteq \mathbf{C}$, $\dim_{\mathrm{LZ}}(X) \leq \dim_{\mathrm{LZ}}(Y)$.

The following theorem states that the Lempel-Ziv dimension of any set $X \subseteq \mathbf{C}$ is completely determined by the dimension of the individual sequences in the set.

Theorem 4. *For all $X \subseteq \mathbf{C}$,*

$$\dim_{\mathrm{LZ}}(X) = \sup_{S \in X} \dim_{\mathrm{LZ}}(S).$$

This theorem implies one important property of dimension, its *countable stability.*

Corollary 1.

1. *For all sets $X, Y \subseteq \mathbf{C}$,*

$$\dim_{\mathrm{LZ}}(X \cup Y) = \max\{\dim_{\mathrm{LZ}}(X), \dim_{\mathrm{LZ}}(Y)\}.$$

2. *Let $X_1, X_2 \dots \subseteq \mathbf{C}$,*

$$\dim_{\mathrm{LZ}}(\bigcup_{i=1}^{\infty} X_i) = \sup_{i \in \mathbb{N}} \dim_{\mathrm{LZ}}(X_i).$$

The main theorem of this section gives an exact characterization of the Lempel-Ziv dimension of an infinite sequence in terms of the asymptotical compression ratio attained by the Lempel-Ziv algorithm.

Theorem 5. *Let $S \in \mathbf{C}$,*

$$\dim_{\mathrm{LZ}}(S) = \liminf_n \frac{|\mathrm{LZ}(S[0 \dots n-1])|}{n}.$$

The following result is a consequence of Remark 1 and Proposition 4. By Theorem 5, the second part is a reformulation of Propositions 2 and 3 in terms of dimension.

Theorem 6. *Let $X \subseteq \mathbf{C}$,*

$$\dim_{\mathrm{p}}(X) \leq \dim_{\mathrm{LZ}}(X) \leq \dim_{\mathrm{FS}}(X).$$

In particular, for all $S \in \mathbf{C}$,

$$\dim_{\mathrm{p}}(S) \leq \dim_{\mathrm{LZ}}(S) \leq \dim_{\mathrm{FS}}(S).$$

By Proposition 1 and Theorem 6 we have the following relationship between the Lempel-Ziv dimension of S and wS.

Theorem 7. *Let $S \in \mathbf{C}$ and $w \in \{0,1\}^*$, then*

$$|\dim_{\mathrm{LZ}}(S) - \dim_{\mathrm{LZ}}(wS)| \leq \dim_{\mathrm{FS}}(S) - \dim_{\mathrm{p}}(S).$$

In particular, if $\dim_{\mathrm{FS}}(S) = \dim_{\mathrm{p}}(S)$, then for all $w \in \{0,1\}^*$,

$$\dim_{\mathrm{LZ}}(S) = \dim_{\mathrm{LZ}}(wS).$$

A consequence of this Theorem is that for sequences such that the polynomial-time dimension and finite-state dimension are equal, the one-bit catastrophe (defined in the next section) is not verified.

Definition 6. *Let the class LZBIT be the set of all sequences S such that for all $w \in \{0,1\}^*$, $\dim_{\mathrm{LZ}}(S) = \dim_{\mathrm{LZ}}(wS)$.*

We use Lempel-Ziv dimension to endow LZBIT with internal dimension structure.

Definition 7. *For $X \subseteq \mathbf{C}$, the* dimension of X *in LZBIT is*

$$\dim(X \mid \mathrm{LZBIT}) = \dim_{\mathrm{LZ}}(X \cap \mathrm{LZBIT}).$$

5 Applications

In this section we partially solve the open question raised by Jack Lutz and other authors [6,7] on the one-bit catastrophe (without the need of studying the asymptotic valid distinct parsing). We also give some results about the Lempel-Ziv compressibility of sequences from results in polynomial-time and finite-state dimension and vice versa.

The one-bit catastrophe conjecture says that the compression ratio of an infinite sequence can change substantially when we add an initial bit on the sequence. In terms of dimension the conjecture reads as: $S \in \mathbf{C}$ verifies the one-bit catastrophe iff $\dim_{\mathrm{LZ}}(S) \neq \dim_{\mathrm{LZ}}(bS)$ for some $b \in \{0,1\}$. Let us illustrate this conjecture with the following example.

Example 1. Let $S = 101100111000\dots1^n0^n\dots \in \mathbf{C}$ that is clearly highly compressed by Lempel-Ziv algorithm. Let $t(w)$ be the number of phrases in the valid distinct parsing of $w \in \{0,1\}^*$. Let $w_1 = S[0\dots29]$, $w_2 = S[0\dots109]$ and $w_3 = S[0\dots239]$. Then,

$$\begin{array}{llll} t(w_1) = 10 & \Rightarrow |\mathrm{LZ}(w_1)| = 35 & t(1w_1) = 13 & \Rightarrow |\mathrm{LZ}(1w_1)| = 53 \\ t(w_2) = 20 & \Rightarrow |\mathrm{LZ}(w_2)| = 101 & t(1w_2) = 29 & \Rightarrow |\mathrm{LZ}(1w_2)| = 146 \\ t(w_3) = 30 & \Rightarrow |\mathrm{LZ}(w_3)| = 151 & t(1w_3) = 45 & \Rightarrow |\mathrm{LZ}(1w_3)| = 271 \end{array}$$

In the case of S it seems that the longer the prefixes are, the better they are compressed. However, in the case of the prefixes of $1S$, it seems that LZ78 does not compress anything.

With results derived in this section, we will show that both sequences are asymptotically highly compressible and do not verify the one-bit catastrophe.

5.1 Stochastic Sequences and Ergodic Measures

In this subsection we present classes of stochastic sequences that do not verify the one-bit catastrophe with probability 1 and determine its Hausdorff, polynomial-time and Lempel-Ziv dimension.

Definition 8. *Let $S \in \mathbf{C}$ and $m \in \mathbb{N}$. Let the* relative probability *for each $w \in \{0,1\}^m$ and each $n \geq m$ be*

$$p_m(w|S[0\dots n-1]) = \frac{\#\{0 \leq i \leq n+m \mid S[i\dots i+m-1] = w\}}{n-m+1}.$$

The limiting relative probability *is defined as*

$$p_m(w|S) = \lim_n p_m(w|S[0\dots n-1]),$$

provided the limit exists.

Definition 9. *A sequence $S \in \mathbf{C}$ is* stochastic *if $p_m(w|S)$ exists for any $m \in \mathbb{N}$ and any $w \in \{0,1\}^m$. Let $\mathcal{S} \subseteq \mathbf{C}$ denote the set of all stochastic sequences.*

Every stochastic sequence $S \in \mathcal{S}$ induces a unique stationary measure on $\mathbf{C}$, $\mu_S : \{0,1\}^* \to [0,1]$ such that $\mu_S(w) = p_{|w|}(w|S)$. (See [1]).

Definition 10. *Let μ be a stationary measure. The set of* frequency typical sequences *of μ is:*

$$\mathcal{T}(\mu) = \{S \in \mathcal{S} \mid \mu_S = \mu\}.$$

Remark 4. Let $S \in \mathcal{S}$ and $w \in \{0,1\}^*$, then $wS \in \mathcal{S}$ and $wS \in \mathcal{T}(\mu_S)$.

The next results will be useful to determine the Lempel-Ziv dimension of frequency typical sequences of a stationary ergodic measure.

Theorem 8. *(LZ78 Universality Theorem) Let μ be a stationary ergodic measure on $\mathbf{C}$ with entropy rate h and $S \in \mathcal{T}(\mu)$ then,*

$$\lim_n \frac{|\mathrm{LZ}(S[0\dots n-1])|}{n} = h \qquad \textit{almost surely.}$$

From [2] we have the following result.

Proposition 5. *Let μ be a stationary ergodic measure with entropy rate h. For all $S \in \mathcal{T}(\mu)$,*

$$\dim_{\mathrm{FS}}(S) = h.$$

Theorem 9. *[5] Let μ be a stationary ergodic measure with entropy rate h. The Hausdorff dimension of the set of frequency typical sequences of μ is h. That is,*

$$\dim_H(\mathcal{T}(\mu)) = h.$$

Corollary 2. *Let μ be a stationary ergodic measure with entropy rate h, then*

1. $\dim_{\mathrm{p}}(\mathcal{T}(\mu)) = \dim_{\mathrm{LZ}}(\mathcal{T}(\mu)) = h$.
2. $\dim(\mathcal{T}(\mu) \,|\, \mathrm{LZBIT})) = h$.
3. *If $S \in \mathcal{T}(\mu)$, then* $\dim_{\mathrm{LZ}}(S) = h$ *almost surely.*
4. *If $S \in \mathcal{T}(\mu)$, then* $S \in \mathrm{LZBIT}$ *almost surely.*

This corollary states that: (i) almost all frequency typical sequences of a stationary ergodic measure have Lempel-Ziv compression ratio equal to the entropy rate; (ii) the compression ratio of all of them is always less than h; and (iii) almost all of them do not verify the one-bit catastrophe.

We use next the last Corollary in a particular class of measures, which includes the uniform measure.

Definition 11. *Let $\alpha \in [0,1]$. The* α-coin-toss probability measure *on* $\mathbf{C}$ *is*

$$\mu^\alpha(w) = (1-\alpha)^{\#(0,w)}\alpha^{\#(1,w)},$$

where $\#(b,w)$ is the number of times that the bit b appears in the string w.

In other words, $\mu^\alpha(w)$ is the probability that $S \in C_w$ when $S \in \mathbf{C}$ is chosen according to a random experiment in which the i^{th} bit of S is decided by tossing a 0/1-valued coin whose probability of 1 is α.

Proposition 6. *Let $\alpha \in [0,1]$ and let $\mathcal{H}$ be the* binary entropy function $\mathcal{H} : [0,1] \to [0,1]$ *defined by*

$$\mathcal{H}(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha)\log\frac{1}{1-\alpha}.$$

Then, μ^α is a stationary ergodic measure with entropy rate $\mathcal{H}(\alpha)$.

Definition 12. *A sequence $S \in \mathbf{C}$ is normal ($S \in$ NORMAL) if $S \in \mathcal{T}(\mu^{\frac{1}{2}})$. That is, S is normal if every string $w \in \{0,1\}^*$ has asymptotic frequency $2^{-|w|}$ in S.*

Theorem 10. *Let $\alpha \in [0,1]$.*

1. $\dim_{\mathrm{H}}(\mathcal{T}(\mu^\alpha)) = \dim_{\mathrm{p}}(\mathcal{T}(\mu^\alpha)) = \dim_{\mathrm{LZ}}(\mathcal{T}(\mu^\alpha)) = \mathcal{H}(\alpha)$.
2. $\dim(\mathcal{T}(\mu^\alpha) \,|\, \mathrm{LZBIT}) = \mathcal{H}(\alpha)$.

3. *For $S \in \mathcal{T}(\mu)$, then* $\dim_{\mathrm{LZ}}(S) = \mathcal{H}(\alpha)$ *almost surely.*
4. *For $S \in \mathcal{T}(\mu)$, then $S \in$ LZBIT almost surely.*

In particular,

1. $\dim_{\mathrm{H}}(\mathrm{NORMAL}) = \dim_{\mathrm{p}}(\mathrm{NORMAL}) = \dim_{\mathrm{LZ}}(\mathrm{NORMAL}) = 1$
2. $\dim(\mathrm{NORMAL} \mid \mathrm{LZBIT}) = 1$.
3. *For $S \in$ NORMAL, then* $\dim_{\mathrm{LZ}}(S) = 1$ *almost surely.*
4. *For $S \in$ NORMAL, then $S \in$LZBIT almost surely.*

In [7] it is proved that sequences with Lempel-Ziv dimension 1 (that is, sequences that Lempel-Ziv algorithm does not compress) are in NORMAL. However, it is also showed that the converse does not hold. Notice that here we show that almost all normal sequences have $\dim_{\mathrm{LZ}}(S) = 1$ and do not verify the one-bit catastrophe.

Since the binary entropy function is surjective on [0,1], then we have that there exist sequences with Lempel-Ziv compression ratio η (for any $\eta \in [0,1]$) that don't verify the one-bit catastrophe:

Corollary 3. *For all $\eta \in [0,1]$ there exist sequences $S \in$ LZBIT with* $\dim_{\mathrm{LZ}}(S) = \eta$.

5.2 Highly LZ Compressible Sequences

Definition 13. *An infinite sequence $S \in \mathbf{C}$ is highly LZ compressible if*

$$\liminf_{n} \frac{|\mathrm{LZ}(S[0 \ldots n-1])|}{n} = 0$$

That is, in terms of Lempel-Ziv dimension, $\dim_{\mathrm{LZ}}(S) = 0$.

Definition 14. *Let $S \in \mathbf{C}$ and $m \in \mathbb{Z}^+$,*

1. *The* factor set $F_m(S)$ *is the set of all finite strings of length m that appear in S.*
2. *The* factor complexity function, $p_S : \mathbb{N} \to \mathbb{N}$, *counts the number of factors of each m, that is $p_S(m) = |F_m(S)|$.*

In [2] it is proved that finite-state dimension of sequences with $\mathrm{p}_S(m) = 2^{o(m)}$ are equal to zero. Then we have the following theorem.

Theorem 11. *Every $S \in \mathbf{C}$ with $\mathrm{p}_S(m) = 2^{o(m)}$ is highly LZ compressible and $S \in$ LZBIT.*

In particular, by using this result on example 1, sequence S is highly compressible and $S \in$ LZBIT since S satisfies $\mathrm{p}_S(m) = m(m+1)$.

Other applications of this Theorem are,

Corollary 4. *1. If S is the binary expansion of a rational number, S is highly LZ compressible and $S \in$ LZBIT.*

2. *Sturmian sequences, Morphic sequences, Automatic sequences are highly LZ compressible and are in LZBIT (See [2]).*
3. *Every $S \in REG$ is highly LZ compressible and $S \in LZBIT$.*

References

1. P. Billingsley. *Probability and Measure.* John Wiley & Sons, Inc., New York, N.Y., 1979.
2. C. Bourke, J. M. Hitchcock, and N. V.Vinodchandran. Entropy rates and finite-state dimension. *Theoretical Computer Science*, 349, 2004.
3. T.M. Cover and J.A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., New York, N.Y., 1991.
4. J. J. Dai, J. I. Lathrop, J. H. Lutz, and E. Mayordomo. Finite-state dimension. *Theoretical Computer Science*, 310:1–33, 2004.
5. K. Hojo, B. Ryabko, and J. Suzuki. Performance of data compression in terms of hausdorff dimension. *TIEICE: IEICE Transactions on Communications/Electronics/ Information and Systems*, 310:1761–1764, 2001.
6. L.A. Pierce II and P.C. Shields. Sequences incompressible by SLZ (LZW) yet fully compressible by ULZ. *Numbers, Information and Complexity, Kluwer Academic Publishers*, pages 385–390, 2000.
7. J. I. Lathrop and M. J. Strauss. A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. In B. Carpentieri, editor, *Compression and Complexity of Sequences '97*, pages 123–135. IEEE Computer Society Press, 1998.
8. M. López-Valdés and E. Mayordomo. Dimension is compression. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, pages 676–685. Springer-Verlag, 2005.
9. J. H. Lutz. Dimension in complexity classes. *SIAM Journal on Computing*, 32:1236–1259, 2003.
10. J. H. Lutz. The dimensions of individual strings and sequences. *Information and Computation*, 187:49–79, 2003.
11. D. Scheinwald. On the lempel-ziv proof and related topics. In *Proceedings of the IEEE*, volume 82, pages 866–871, 1994.
12. J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transactions on Information Theory*, 24:530–536, 1978.
13. A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25:83–124, 1970.

Characterizing Valiant's Algebraic Complexity Classes

Guillaume Malod[1] and Natacha Portier[2]

[1] Kyoto University, Japan
malod@kuis.kyoto-u.ac.jp
[2] École Normale Supérieure de Lyon, France
Natacha.Portier@ens-lyon.fr

Abstract. Valiant introduced 20 years ago a theory to study the complexity of polynomial families. Using arithmetic circuits as computation model, these classes are easy to define and open to combinatorial techniques. In this paper we gather old and new results under a unifying theme, namely the restrictions imposed upon the gates, building a hierarchy from formulas to circuits. As a consequence we get simpler proofs for known results such as the equality of the classes VNP and VNP_e or the completeness of the determinant for VQP, and new results such as a characterization of the class VP or answers to both a conjecture and a problem raised by Bürgisser [1]. We also show that for circuits of polynomial depth and unbounded size these models have the same expressive power and characterize a uniform version of VNP.

Keywords: Algebraic complexity, Valiant's theory, polynomials, Permanent, Determinant, arithmetic circuits, skew circuits.

1 Introduction

The common case in favor of studying arithmetic circuits is that they offer a compact representation of polynomials. Results by Kaltofen [2] and von zur Gathen [3] show the feasibility of such a representation scheme by studying the effects of standard symbolic manipulations. Valiant's algebraic complexity classes appear in this context as the relevant formalization of intractability, explaining for instance why we have no efficient algorithm for general iterated derivation, as it would imply the equality of the classes VP and VNP. Arithmetic circuits also relate to Boolean complexity. Kabanets and Impagliazzo [4] link the derandomization of Polynomial Identity Testing with super-polynomial arithmetic circuit lower bounds for the Permanent (i.e. the separation of the classes VP and VNP). Koiran [5] shows that the complexity of computing certain integers such as $n!$ is related to Valiant's classes. Arithmetic circuits can also be considered as Boolean inputs and define new problems with interesting consequences in complexity, as is shown by [6], where the problem of deciding whether an arithmetic circuit computes a positive number is related to numerical analysis.

Our interest in Valiant's classes is based on a slightly different perspective. These classes can be seen as representing computations by circuits in general, be they arithmetic or Boolean. Their definition is very simple so that the combinatorial insights are unfettered by computational details. Moreover the reductions used are low level (p-projections, as used for example in [7]), thus retaining the algebraic character of problems and giving very strong completeness results.

We stress this combinatorial aspect by introducing a restriction on circuits to give a characterization of the class VP (Theorem 1). This greatly simplifies one the main steps in the completeness proof of the Permanent. It also illustrates our point on the general nature of these classes, because we can deduce a new circuit characterization of LOGCFL and #LOGCFL (Propositions 2 and 3).

A stronger restriction applies to the class VQP, which captures the complexity of the Determinant. We use it to give full answer to a conjecture by Bürgisser [1] stating that several operations of linear algebra are VQP-complete (Theorem 5). The restricted circuits used have in fact been previously introduced by Toda [8]. We import his definition of a class capturing the complexity of the Determinant and suggest that it is better suited to the task than VQP. The completeness of the Determinant for this class yields an answer to another question raised by Bürgisser: the Determinant family is indeed linearly closed (Proposition 5). We finally use similar circuit techniques to characterize a uniform version of VNP (Theorem 9).

2 Basic Definitions

Valiant's complexity classes revolve around the representation of polynomials over a given field by arithmetic circuits. More details can be found in [1,3].

Definition 1. An *arithmetic circuit* is a finite acyclic directed graph with vertices of in-degree 0 or 2 and exactly one vertex of out-degree 0. Vertices of in-degree 0 are called *inputs* and labeled by a constant or a variable. The other vertices, of in-degree 2, are labeled by $\times$ or $+$ and called computation gates. We distinguish left and right arguments to a computation gate (i.e. each arrow in our graph is implicitly labelled with L or R depending on whether it is a left or right input). The vertex of out-degree 0 is called the *output*. The vertices of a circuit are commonly called *gates* and its edges *arrows*.

The polynomial represented by a circuit can easily be defined by induction. Circuits represent a computation where one can reuse partial results. If we do not allow this, that is if we require each argument to be computed especially for a given computation step, then the graph underlying the circuit must be a tree. Such circuits are called *expressions*, *arithmetic terms* or *formulas* (we shall use the latter). We associate the following parameters to a given circuit.

Definition 2. The *size* of a circuit is its number of gates. The *depth* is the maximal length of a directed path from an input to an output. The *degree* of a gate is defined recursively: any input is of degree 1; the degree of a $+$ gate is the max of the incoming degrees; the degree of a $\times$ gate is the sum of the incoming degrees. The degree of the circuit is the degree of its output gate.

As usual in complexity theory we are interested in asymptotics, in this case the growth of the size of the circuits representing a sequence of polynomials. We give here the definitions of Valiant's classes and the reductions used. Note that the classes depend on a chosen field, but as we are interested in combinatorial techniques this will not play a role in this paper.

Definition 3. A sequence of polynomials (f_n) belongs to VP if there exists a sequence of circuits C_n of polynomially bounded size and degree such that C_n represents f_n.
A sequence of polynomials (f_n) belongs to VNP if there exists a polynomial p and a sequence $g_n \in$ VP such that $f_n(\bar{x}) = \sum_{\bar{\epsilon} \in \{0,1\}^{p(|\bar{x}|)}} g_n(\bar{x}, \bar{\epsilon})$.
A polynomial f is a *projection* of a polynomial g if $f(\bar{x}) = g(a_1, \ldots, a_m)$, where the a_i are elements of the field or variables among $x_1, \ldots, x_n$. A sequence (f_n) is a *p-projection* of a sequence (g_n) if there exists a polynomially bounded function $t(n)$ such that f_n is a projection of $g_{t(n)}$ for all n.

It is obvious that VP is included in VNP. Valiant's hypothesis is that this inclusion is strict; it remains a major open problem of complexity theory. We can define similar classes using formulas in place of circuits. These classes play an important part in the completeness proof of the permanent.

Definition 4. A sequence of polynomials (f_n) belongs to the class $\mathrm{VP_e}$ if there exists a sequence of formulas F_n of polynomially bounded size such that F_n represents f_n.
A sequence of polynomials (f_n) belongs to $\mathrm{VNP_e}$ if there exists a polynomial p and a sequence $g_n \in \mathrm{VP_e}$ such that $f_n(\bar{x}) = \sum_{\bar{\epsilon} \in \{0,1\}^{p(|\bar{x}|)}} g_n(\bar{x}, \bar{\epsilon})$.

The main result in Valiant's theory is the completeness of the Permanent family of polynomials for the class VNP, over fields of characteristic different from 2. The permanent of a matrix of size n with variables entries $z_{i,j}$ is defined as $\mathrm{PER}_n(z_{i,j}) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} z_{i,\sigma(i)}$. In this definition, S_n is the group of permutations of $\{1, \ldots, n\}$. This result stands in stark contrast to the fact that the Determinant family belongs to the class VP. The determinant is defined as the permanent but with positive and negative monomials depending on the sign $s(\sigma)$ of the permutation: $\mathrm{DET}_n(z_{i,j}) = \sum_{\sigma \in S_n} s(\sigma) \prod_{i=1}^{n} z_{i,\sigma(i)}$.

3 Characterizing VP

Whereas the class VNP captures the complexity of the Permanent and many other problems, there is no natural complete problem for the class VP. We give here an intuitive characterization which we hope may provide better insight. For this purpose we introduce the following definition, exploiting the interplay between circuits and formulas in Valiant's theory.

Definition 5. Let α be a gate receiving arrows from gates β and γ. We say that α is *disjoint* if the sub-circuits associated to β and γ are disjoint from one another. A circuit is *multiplicatively disjoint* if all its multiplication gates are disjoint.

A circuit is a formula if and only if all its gates are disjoint. A multiplicatively disjoint circuit behaves like a formula for multiplications. Disjoint multiplications can be seen as a way to control the degree of the polynomial computed by a circuit, which links this technique to the retarded multiplication scheme used in [9] to characterize the class $\sharp$P. However it also provides combinatorial information, as we will see in the next section. Multiplicatively disjoint circuits of polynomial size characterize VP.

Theorem 1. *A sequence of polynomials (f_n) belongs to* VP *if and only if there exists a sequence (C_n) of multiplicatively disjoint circuits, of polynomially bounded size, such that C_n represents the polynomial f_n.*

This theorem is an obvious consequence of Lemma 1 and 2 below. The first lemma can be shown by an easy induction on the size of the circuit. For the second, the basic idea is comparable to the naïve transformation of a circuit into a formula by duplicating gates. However, for multiplicatively disjoint circuit, the construction can be done in such a way that the degree of the circuit is a bound on the number of copies needed for a given gate, so that we avoid a potentially exponential growth in size. A proof of this theorem can also be obtained from the characterization of circuits of polynomial size and degree by semi-unbounded circuits of polynomial size and logarithmic depth in [10].

Lemma 1. *If C is a multiplicatively disjoint circuit of size t, its degree is less than or equal to t.*

Lemma 2. *If C is a circuit of size t and degree d, there exists a multiplicatively disjoint circuit C', which computes the same polynomial and whose size is less than or equal to dt.*

4 Consequences

4.1 Formulas and Circuits

One major open question is whether circuits are more powerful than formulas at the polynomial level, i.e. whether the inclusion $\mathrm{VP_e} \subseteq \mathrm{VP}$ is strict or not. The first step of the completeness proof of the Permanent is to show that under a Boolean sum formulas and circuits have the same power. A technically involved proof of this can be found for example in [1]. The above characterization of VP yields a simpler and more intuitive proof, which we will sketch here.

Theorem 2. $\mathrm{VNP} = \mathrm{VNP_e}$ *over any field.*

The inclusion $\mathrm{VNP_e} \subseteq \mathrm{VNP}$ is obvious. It is easy to see that, in order to prove the inclusion $\mathrm{VNP} \subseteq \mathrm{VNP_e}$, we need only prove the inclusion $\mathrm{VP} \subseteq \mathrm{VNP_e}$. We therefore need to express the polynomial represented by a circuit as a sum of formulas. For a given circuit we will consider graphs called *parse trees*. These graphs appear under different names in several previous works [10,11,12,13,14].

We will use them in the context of arithmetic circuits, in the spirit of this quote from [11]: a parse tree is "a family tree which charts the generation of a particular monomial in the final result".

Definition 6. The set of parse trees of a circuit C is defined by induction on its size:

- If C is of size 1 it has only one parse tree, itself.
- If the output gate of C is a $+$ gate whose arguments are the gates α and β, the parse trees of C are obtained by taking either a parse tree of C_α and the arrow from α to the output or a parse tree of C_β and the arrow from β to the output.
- If the output gate of C is a $\times$ gate whose arguments are the gates α and β, the parse trees of C are obtained by taking a parse tree of C_α and a parse tree of a disjoint copy of C_β and the arrows from α and β to the output.

It turns out that the polynomial computed by a circuit is the sum of the values of its parse trees. This is true in general, and can easily be shown by induction. We write $\mathrm{PT}(C)$ for the set of parse trees of a circuit C and $\mathrm{val}(T)$ for the value of parse tree T.

Lemma 3. *If C is a circuit for the polynomial f, $f(\bar{x}) = \sum_{T \in \mathrm{PT}(C)} \mathrm{val}(T)$.*

To prove the inclusion $\mathrm{VP} \subseteq \mathrm{VNP_e}$ we thus wish to write a polynomial in VP as a sum of formulas. We can use the previous lemma, but we need to show that we can indeed sum over all parse trees and compute the value of a parse tree. In other words we will in fact sum over all possible Boolean words of a given length, as in the definition of $\mathrm{VNP_e}$, therefore we need to have a formula to recognize when a word encodes a parse tree and to compute its value. This task is easier for multiplicatively disjoint circuits, thanks to the following proposition, which is not hard to prove.

Proposition 1. *A circuit C is multiplicatively disjoint iff each parse tree of C is a sub-graph of C.*

The useful implication for us is that in the case of multiplicatively disjoint circuits all parse trees are sub-graphs. And since the circuit is of polynomial size, it is straightforward, if somewhat tedious, to recognize and compute the value of the parse trees of a circuit with a formula.

4.2 Boolean Classes Defined by Multiplicatively Disjoint Circuits

In this short subsection, we illustrate the general nature of results in Valiant's theory by applying them in the Boolean setting. The name VP might suggest that the related Boolean class is P, but if one looks at circuit definitions of Boolean classes, it is obvious that the closest class is LOGCFL. This is reflected in the *polynomial proof tree size* property identified by Venkateswaran [15] to characterize LOGCFL. In our context, this property is illustrated by the fact that all proof trees of a multiplicatively disjoint circuit are sub-circuits, and thus have the same size bound as the circuit. We get a straightforward characterization of LOGCFL by multiplicatively disjoint circuits, where we use the Boolean operators as ring operations and define the degree of a circuit accordingly. The proof uses Venkateswaran's characterization of LOGCFL by semi-unbounded circuits of polynomial size and logarithmic depth in [15], then the equivalence with circuits of polynomial size and degree from [10], and finally the equivalence with multiplicatively disjoint circuits of polynomial size from Theorem 1.

Proposition 2. LOGCFL *is the class of languages accepted by uniform sequences of multiplicatively disjoint Boolean circuits of polynomial size.*

By arithmetizing the circuit definition of this class in the manner described in [16] we get a characterization of #LOGCFL.

Proposition 3. #LOGCFL *is the class of functions computed by uniform sequences of multiplicatively disjoint arithmetic circuits of polynomial size.*

Note that these results can be contrasted to the known links between NC_1 and formulas, and between NL and skew circuits: uniform sequences of formulas can be used to characterize NC_1 and $\sharp NC_1$ while skew circuits characterize NL or $\sharp L$. The hierarchy of restrictions we are studying in this paper thus also exists in the Boolean case.

5 The Complexity of the Determinant

5.1 The Class VQP

The Determinant family is known to belong to the class VP. However it is not known to be VP-complete, nor is it thought to be. The class VQP, defined via circuits of quasi-polynomial size, was introduced to further study the complexity of the Determinant. Indeed one can find proofs of completeness of the Determinant for VQP in [1,3]. Strengthening the restriction on multiplications enabled us to give in [14] a simpler proof. As a matter of fact, this strengthened restriction had already been introduced by Toda in [8] with the definition of *weakly skew circuits*. This last work is extremely relevant to the complexity of the Determinant in Valiant's theory: the connection between skew or weakly skew circuits and the determinant of integer matrices is well known, but it is surprising that the class VQP is used to capture the complexity in Valiant's setting when a definition based on skew circuits is much more natural. We define such a class in the next section and provide details there. As the techniques for both classes are similar, and as we strongly favor this new class, only definitions and results will be stated for the class VQP, without further details.

Definition 7. A function t from $\mathbb{N}$ to $\mathbb{N}$ is *quasi-polynomially bounded* if there exist two constants a and b such that $t(n) \leq n^{a \cdot \log^b n}$ for all $n \geq 2$.

A sequence of polynomials (f_n) belongs to the class VQP if its number of variables and degree is polynomially bounded and if it is represented by a circuit of quasi-polynomially bounded size.
A sequence (f_n) is a *qp-projection* of a sequence (g_n) if there exists a quasi-polynomially bounded function t such that for all n f_n is a projection of $g_{t(n)}$

Theorem 3. *The Determinant is* VQP*-complete over any field.*

The following algebraic characterization of whether VNP is included in VQP is noted in [3] (it is shown in [1] that VQP is not included in VNP).

Theorem 4. VNP $\subseteq$ VQP *iff the Permanent is a qp-projection of the Determinant.*

Consider now the families of polynomials (F_n), (G_n) and (H_n) defined by $F_n = \mathrm{Tr}(X^n)$, $G_n = \mathrm{Tr}(X_1 \cdots X_n)$ and $H_n = \mathrm{Tr}(\mathrm{DET}(X) \cdot X^{-1})$, where Tr is the trace, and X or X_i are matrices with n^2 variables. The following theorem, which can be proved using the ideas in the next section, is our answer to conjecture 8.1 from [1][1].

Theorem 5. *The families* (F_n)*,* (G_n) *and* (H_n) *are* VQP*-complete over any field.*

5.2 The Class $\mathrm{VP}_{\mathrm{ws}}$

We have already said that [8] gives an excellent account of the complexity of the Determinant, which can be immediately transposed into Valiant's setting. In fact, Toda defines the very natural class DET(poly) of polynomial families which can be expressed as the determinant of a sequence of matrices (with variable or constant entries) of polynomially bounded size. This class is shown to be characterized by skew arithmetic circuits, and equivalently by a new type of circuits, called weakly skew. Recall that a circuit is *skew* if all multiplication gates have at most one argument which is not an input gate. The condition is somewhat relaxed for weakly skew circuits.

Definition 8. A circuit is *weakly skew* if for any multiplication gate α, receiving arrows from gates β and γ, one of the two sub-circuits C_β or C_γ is only connected to the rest of the circuit by the arrow going to α.

One can see formulas as computations where each argument of a gate is computed exclusively for that gate. In the case of weakly skew circuits, this condition must hold for at least one of the two arguments of each multiplication. This requirement will be sufficient for a an important construction to go through for weakly skew circuits: the so-called universality property. It has already been proved for formulas and the Determinant: the polynomial computed by a formula s is a projection of the Determinant of a matrix of size polynomial in s. The

[1] A partial answer to this conjecture was given in [17].

main step of the proof is to build weighted graphs with adequate weight. Let G be an edge-weighted directed graph with two vertices s and t, the weight of a path from s to t is the product of the weights of the edges appearing in the path. The weight of (s,t) in G is the sum of the weights of all paths from s to t. Such a graph can also be built for weakly skew circuits.

Lemma 4. *Let C be a weakly skew circuit of size m, there exists an acyclic directed graph G, with two distinguished vertices s and t, such that: G is of size $m+1$ and the weight of (s,t) in G is the polynomial computed by C.*

When proving the lemma for formulas, one can build the graph by induction in the following manner: an input gate becomes an edge weighted with the corresponding variable or constant; for an addition gate we place the graphs corresponding to the arguments in parallel; for a multiplication gate we place them in series. Going from a formula to a weakly skew circuit one must strengthen the property being proved so that it applies to circuits with multiple output gates. Therefore we wish to build a graph which has a vertex t_α for several gates α in the circuit such that the weight of the graph between s and t_α is the polynomial represented by α. The induction steps above still work except that by placing the circuits in series when we multiply we may change the weight of the gates of the second circuit. The weakly skew condition guarantees that we will not need the values of these gates later in the circuit, so that the construction goes through. Indeed, one can see weakly skew circuits as the most expressive circuits for which this construction can work, in the sense that any polynomial which is the weight of a graph of size s can be computed by a weakly skew circuit of size polynomial in s (this is a consequence of the completeness results which follow). From this construction we can show the universality of the Determinant for weakly skew circuits, as noted by Toda.

Lemma 5. *If f is a polynomial computable by a weakly skew circuit of size m, f is a projection of* DET_{m+1}.

Let us now transplant Toda's class in Valiant's framework: we will call it $\mathrm{VP}_{\mathrm{ws}}$ and define it directly by weakly skew circuits.

Definition 9. A sequence of polynomials (f_n) belongs to the class $\mathrm{VP}_{\mathrm{ws}}$ if it is represented by a sequence of weakly skew circuits of polynomially bounded size.

The previous universality lemma, together with a computation by weakly skew circuits (see the combinatorial work of Mahajan & Vinay [18]), gives us a natural proof of the completeness of the Determinant for the class $\mathrm{VP}_{\mathrm{ws}}$, a direct transposition of Toda's work in Valiant's setting. Note that this completeness is under standard p-projections, which is one reason we suggest this class be preferred to VQP.

Theorem 6. *The sequence* (DET_n) *is* $\mathrm{VP}_{\mathrm{ws}}$*-complete over any field.*

Defining $\mathrm{VP}_{\mathrm{ws}}$ via weakly skew circuits puts it naturally between VP_{e} and VP. Using Lemma 4 and some graph tricks, we can also show the completeness of the families (F_n), (G_n) and (H_n) for this new class. The completeness of (F_n) then implies an alternative characterization of $\mathrm{VP}_{\mathrm{ws}}$ by skew circuits.

Theorem 7. *The families (F_n), (G_n) and (H_n) are $\mathrm{VP}_{\mathrm{ws}}$-complete over any field.*

Proposition 4. *A sequence of polynomials (f_n) belongs to the class $\mathrm{VP}_{\mathrm{ws}}$ if it is represented by a sequence of skew circuits of polynomially bounded size.*

5.3 The Permanent and the Determinant

Introducing $\mathrm{VP}_{\mathrm{ws}}$ enables us to provide another complexity theoretic characterization of the relation between the Permanent and the Determinant, similar to Theorem 4, but more natural. Both these results are interesting in that they relate a question from computational complexity to an easily stated mathematical problem.

Theorem 8. *The Permanent is a p-projection of the Determinant iff* $\mathrm{VP}_{\mathrm{ws}} = \mathrm{VNP}$.

Several articles have been written on the links between the Permanent and the Determinant, going back to Pólya [19], who asks whether one can change the sign of the entries of a $\{0, 1\}$ matrix so that the Determinant of the resulting matrix is the Permanent of the original one. A result such as the one above indicates that even the more general procedure of computing a Permanent as the Determinant of a polynomially bigger matrix is probably not possible in the general case.

Using the completeness of the Determinant for the class $\mathrm{VP}_{\mathrm{ws}}$, we can also answer a question raised by Bürgisser [1] (Problem 3.2). He defines the notion of *linearly closed families.*

Definition 10. A family (g_n) is called *linearly closed* if any linear combination $\sum_{k=1}^{n} \lambda_k g_{i_k}$ is a projection of some g_m, where m is polynomially bounded in the number n of terms and $\max_k i_k$. Hereby the sets of variables of the g_i are supposed to be (made) disjoint for distinct k.

Bürgisser then asks whether the Determinant family has this property . The answer is positive. It is interesting to see this "mathematical" property of the Determinant proved by a technique from complexity theory.

Proposition 5. *The Determinant is linearly closed.*

Indeed, each Determinant in the linear combination can be computed by a weakly skew circuit. If we multiply the result of each circuit by the appropriate coefficient and then sum them, the resulting circuit is still weakly skew. By completeness, this circuit can be computed as the Determinant of a matrix whose size depends on the size and number of matrices in the linear combination.

6 Characterizing Uniform VNP

We have considered the increasing expressive power of the following sequence of models, when the size is polynomially bounded: formulas, weakly skew circuits, multiplicatively disjoint circuits. One of the reasons VQP was considered a "good" class is that if we allow a quasi-polynomially bounded size, all these classes are equal (cf. [3]). This collapse also occurs if we polynomially bound the depth rather than the size. And, as we shall see in this section, in the uniform case the resulting class is VNP.

We wish to compare the respective expressive power of Boolean sums in front of a circuit of polynomial size and degree (VNP) on the one hand and of circuits of polynomial depth and degree on the other. This is related to the characterization of ♯P via circuits of polynomial depth and degree in [13]. We will show that a similar theorem holds for a uniform version of Valiant's algebraic classes.

At the non-uniform level it is easy to see that circuits of polynomial depth and degree are at least as powerful as VNP. Indeed a sequence in VNP is defined from a sequence in VP which is represented by circuits of polynomial size and degree, and therefore polynomial depth and degree. By computing in parallel all the values of these circuits for all Boolean strings of appropriate length and then summing, we get a circuit of polynomial depth and degree. The summation can be done in polynomial depth because there is a simply exponential number of gates to sum.

For the converse we would like to express the polynomial computed by a circuit of polynomial depth and degree as a sum of the values of a circuit of polynomial size and degree. We will use the same strategy as when proving the equality of VNP and $\mathrm{VNP_e}$. The value of a circuit is written as the sum of the values of its parse trees. Although there is no polynomial bound on the size of our original circuit, the constraints on depth and degree give us a constraint on the size of the parse trees, as noticed in [13].

Lemma 6. *Any parse tree of a circuit of depth p and degree d is of size less than or equal to pd.*

However we also need to recognize efficiently whether a Boolean string encodes a parse tree or not (previously we used the sub-circuit property because our circuits were of polynomial size). This will be made possible by the second ingredient, uniformity. We will use the condition given in [13]. Define the *direct connection language* of a sequence of circuits C_n as the set of strings of the form $\langle n, g, y, p\rangle$ such that either (i) g is an addition gate in c_n and y is an input of g, or (ii) g is a multiplication gate in c_n and y is a left or right input of g depending on p, or (iii) g is a gate name in c_n and y is the type of g. A sequence of circuits C_n is DLOGTIME-uniform if its direct connection language can be recognized by a deterministic Turing machine in time logarithmic in the size of the circuits. In our case, with circuits of exponential size, it means that we can get information on an arrow or a gate in polynomial time.

Let us now define the uniform classes we have mentioned. For Valiant's classes, P-uniformity is the most plausible notion, meaning that the circuit C_n is produced by a Turing machine in polynomial time upon input of n in unary. We will also assume that the circuits in the sequence use a fixed set of constants.

Definition 11. A sequence of polynomials is in the class $\mathrm{VP_u}$ if it is represented by a P-uniform sequence of circuits of polynomial size and degree.
A sequence of polynomials (f_n) belongs to $\mathrm{VNP_u}$ if there exists a polynomial p and a sequence $g_n \in \mathrm{VP_u}$ such that $f_n(\bar{x}) = \sum_{\bar{\epsilon}\in\{0,1\}^{p(|\bar{x}|)}} g_n(\bar{x},\bar{\epsilon})$.

Theorem 9. *A sequence of polynomials* (f_n) *belongs to* $\mathrm{VNP_u}$ *iff it can be represented by a* DLOGTIME*-uniform sequence of circuits of polynomial depth and degree.*

The proof of this theorem follows the sketch given above. The remainder is technical details which we will not give here. Note the similarity of this characterization with the characterization of ♯P by Venkateswaran [13]. In both cases the class characterized is uniform. In our description of the proof strategy we emphasize the role played by uniformity. What happens in the non-uniform case?

We will use a converse of Valiant's criterion (cf. [1]) to answer this question. Valiant gave a criterion for showing that specific sequences of polynomials belong to VNP, the rough idea being that sequences whose coefficient function is in ♯P/*poly* belong to VNP. One can show a converse of this theorem by using the coefficient function (we will not give details here, this is noted in [20] and can be proved using techniques from [14]). Such a converse states that if we have a family of functions (f_n), where $f_n : \{0,1\}^n \to \left[0,\ldots,2^{p(n)}\right]$, and if we define $g_n(x_1,\ldots,x_n) = \sum_{\bar{\epsilon}\in\{0,1\}^n} f_n(\bar{\epsilon})\ x_1^{\epsilon_1}\cdots x_n^{\epsilon_n}$, then $(g_n) \in \mathrm{VNP}$ implies $(f_n) \in$ ♯P/*poly*.

Now consider any sequence of functions $f_n : \{0,1\}^n \to \left[0,\ldots,2^{p(n)}\right]$ and view it as the coefficient function of a sequence $g_n(\bar{x}) = \sum_{\bar{\epsilon}} f_n(\bar{\epsilon})x_1^{\epsilon_1}\cdots x_n^{\epsilon_n}$. We can compute in polynomial depth these monomials and the integer coefficients and just sum them, so that the sequence (g_n) can be computed by a sequence of circuits of polynomial depth and degree. If the hypothesis is true in this non-uniform case, then (g_n) belongs to VNP. Thus (f_n) belongs to ♯P/*poly*. However there exists functions which are not in PSPACE/*poly*, and thus not in ♯P/*poly*, which contradicts the assumption. Thus the uniformity condition is not insignificant, but rather an essential ingredient of the proof.

7 Conclusion

We have shown in this paper that different classes in Valiant's framework can be defined via a hierarchy of circuits of polynomial size, from formulas to weakly skew circuits to multiplicatively disjoint circuits, and that all these restrictions become equivalent for polynomial depth and (in the uniform case) define the class VNP. These characterizations came with new results and new proofs of old results. In our view, one important aim of this paper is to bring attention to the work of Toda [8] and suggest the adoption of the class $\mathrm{VP_{ws}}$.

To stress the importance of this class we could like to find other complete polynomials. For any polynomial family which is shown to be in VP we should check if one can show that it is $\mathrm{VP_{ws}}$-complete. The class $\mathrm{VP_{ws}}$ can also be seen as capturing the computational power of directed acyclic graphs with weights. When one builds a graph as in Lemma 4 starting from a formula, the resulting directed graph is *series-parallel*, a property which has been studied in the context of task ordering or parametrized complexity. The question of the respective power of weakly skew circuits versus formulas, which is the question of whether the Determinant can be computed by formulas, is exactly the problem of whether a general st-dag can be transformed into a series-parallel one of same weight without an explosion of its size. We think it would be interesting to study these links.

As for the characterization of VP, it could help us find a natural complete problem. A good strategy for this is to look more closely at the class LOGCFL, its properties and complete problems. One possibility would be to use the tensor formulas defined in [21], where an example of a LOGCFL-complete problem is given. Indeed, the different kinds of tensor formulas seem to fit very well with the classes $\mathrm{VP_{ws}}$, VP and VNP, and we hope to explore this link in a future work.

References

1. Bürgisser, P.: Completeness and reduction in algebraic complexity theory. Volume 7 of Algorithms and Computation in Mathematics. Springer-Verlag, Berlin (2000)
2. Kaltofen, E.: Uniform closure properties of p-computable functions. In: STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1986) 330–337
3. von zur Gathen, J.: Feasible arithmetic computations: Valiant's hypothesis. J. Symb. Comput. **4** (1987) 137–172
4. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA. (2003) 355–364
5. Koiran, P.: Valiant's model and the cost of computing integers. Computational Complexity **13** (2005) 131–146
6. Allender, E., Bürgisser, P., Kjeldgaard-Pedersen, J., Miltersen, P.: On the complexity of numerical analysis. Electronic Colloquium on Computational Complexity (ECCC) (2004)
7. Immerman, N., Landau, S.: The complexity of iterated multiplication. Inf. Comput. **116** (1995) 103–116
8. Toda, S.: Classes of arithmetic circuits capturing the complexity of computing the determinant. IEICE Transactions on Information and Systems **E75-D** (1992) 116–124
9. Babai, L., Fortnow, L.: Arithmetization: a new method in structural complexity theory. Comput. Complexity **1** (1991) 41–66
10. Allender, E., Jiao, J., Mahajan, M., Vinay, V.: Non-commutative arithmetic circuits: depth reduction and size lower bounds. Theoret. Comput. Sci. **209** (1998) 47–86

11. Jerrum, M., Snir, M.: Some exact complexity results for straight-line computations over semirings. J. ACM **29** (1982) 874–897
12. Venkateswaran, H., Tompa, M.: A new pebble game that characterizes parallel complexity classes. SIAM J. Comput. **18** (1989) 533–549
13. Venkateswaran, H.: Circuit definitions of nondeterministic complexity classes. SIAM J. Comput. **21** (1992) 655–670
14. Malod, G.: Polynômes et coefficients. PhD thesis, Université Claude Bernard Lyon 1 (2003)
15. Venkateswaran, H.: Properties that characterize LOGCFL. J. Comput. Syst. Sci. **43** (1991) 380–404
16. Allender, E.: Arithmetic Circuits and Counting Complexity Classes. In Krajicek, J., ed.: Complexity of Computations and Proofs. Volume 13 of Quaderni di Matematica. Seconda Universita di Napoli (2004) 33–72
17. Bläser, M.: Complete problems for valiant's class of qp-computable families of polynomials. In: COCOON '01: Proceedings of the 7th Annual International Conference on Computing and Combinatorics, London, UK, Springer-Verlag (2001) 1–10
18. Mahajan, M., Vinay, V.: Determinant: Combinatorics, algorithms, and complexity. Chicago J. Theor. Comput. Sci. **1997** (1997)
19. Pólya, G.: Aufgabe 424. Arch. Math. Phys. **20** (1913) 271
20. Périfel, S.: Polynômes donnés par des circuits algébriques et généralisation du modèle de Valiant. Master's thesis, École Normal Supérieure de Lyon, France (2004)
21. Damm, C., Holzer, M., McKenzie, P.: The complexity of tensor calculus. Computational Complexity **11** (2002) 54–89

The Price of Defense*

Marios Mavronicolas[1], Loizos Michael[2], Vicky Papadopoulou[1], Anna Philippou[1], and Paul Spirakis[3]

[1] Department of Computer Science, University of Cyprus, Nicosia CY-1678, Cyprus
{mavronic,viki,annap}@ucy.ac.cy

[2] Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138
loizos@eecs.harvard.edu

[3] Research Academic Computer Technology Institute (RACTI), Rion, Patras 26500, Greece & Department of Computer Engineering and Informatics, University of Patras, Rion, Patras 26500, Greece
spirakis@cti.gr

Abstract. We consider a *strategic game* with two classes of confronting randomized players on a graph $G(V, E)$: ν *attackers*, each choosing vertices and wishing to minimize the probability of being caught, and a *defender*, who chooses edges and gains the expected number of attackers it catches. The *Price of Defense* is the worst-case ratio, over all Nash equilibria, of the *optimal* gain of the defender over its gain at a Nash equilibrium. We provide a comprehensive collection of trade-offs between the Price of Defense and the computational efficiency of Nash equilibria.

- Through reduction to a *Two-Players, Constant-Sum Game*, we prove that a Nash equilibrium can be computed in polynomial time. The reduction does not provide any apparent guarantees on the Price of Defense.
- To obtain such, we analyze several structured Nash equilibria:
  - In a *Matching Nash equilibrium*, the support of the defender is an *Edge Cover*. We prove that they can be computed in polynomial time, and they incur a Price of Defense of $\alpha(G)$, the *Independence Number* of G.
  - In a *Perfect Matching Nash equilibrium*, the support of the defender is a *Perfect Matching*. We prove that they can be computed in polynomial time, and they incur a Price of Defense of $\frac{|V|}{2}$.
  - In a *Defender Uniform Nash equilibrium*, the defender chooses uniformly each edge in its support. We prove that they incur a Price of Defense falling between those for Matching and Perfect Matching Nash Equilibria; however, it is $\mathcal{NP}$-complete to decide their existence.
  - In an *Attacker Symmetric* and *Uniform Nash equilibrium*, all attackers have a common support on which each uses a uniform distribution. We prove that they can be computed in polynomial time and incur a Price of Defense of either $\frac{|V|}{2}$ or $\alpha(G)$.

* This work was partially supported by the IST Program of the European Union under contract number IST-2004-001907 (DELIS).

1 Introduction

Motivation and Framework. We revisit a network game with *attackers* and a *defender*, introduced recently by Mavronicolas *et al.* [9] and further studied in [5,10]; the game was conceived as an appropriate theoretical model of security *attacks* and *defenses* in emerging networks like the Internet. In this network game, nodes are vulnerable to infection by *threats*, called *attackers*. Available to the network is a security software (or *firewall* [3]), called the *defender*, cleaning some part of the network.

This network game is partially motivated by *Network Edge Security* [8], a new distributed firewall architecture where a firewall is implemented in a *distributed* way and protects the subnetwork spanned by the nodes participating in the distributed implementation. The simplest case where the subnetwork is a single link (with its two incident nodes) offers the initial basis for the theoretical model of Mavronicolas *et al.* [9]. Understanding the mathematical pitfalls of this simplest model is a necessary prerequisite for making rigorous progress in analyzing distributed firewall architectures with more involved topologies.

Each attacker (called *vertex player*) targets a *node* of the network chosen via its own probability distribution on nodes; the defender (called *edge player*) chooses a single *link* via its probability distribution on links. A node chosen by an attacker is destroyed unless it crosses the link being cleaned by the defender. The *Individual Profit* of an attacker is the probability that it is not caught; the *Individual Profit* of the defender is the expected number of attackers it catches.

To the best of our knowledge, this network game is the *first* strategic game where the network is *explicitly* modeled as a non-cooperative player (the defender). Unlike previously studied games that evaluated the effect of selfish behavior on system performance using the *Price of Anarchy* [7] (which *implicitly* modeled the system), we pursue this evaluation via the *Price of Defense*: the *worst-case* ratio of ν over the Individual Profit of the defender.

We are interested in analyzing the Price of Defense for *Nash equilibria* [11,12], where no single player has an incentive to deviate from its strategy. How does the Price of Defense vary with Nash equilibria? Are there Nash equilibria that both are tractable and offer good Price of Defense? Such questions are the focus of our work. Our answers make a comprehensive collection of trade-offs between Price of Defense and computational complexity of Nash equilibria.

Contribution. We prove that a (mixed) Nash equilibrium for our network game can be computed in polynomial time (Theorem 4). The proof is by reduction to the case of two players (one attacker and one defender), which is shown to be *Constant-Sum. Two-Players, Constant-Sum* games are reducible to *Linear Programming* [13], hence solvable in polynomial time. The reduction to Linear Programming hides the Price of Defense. This invites considering classes of Nash equilibria with sufficient structure for evaluating the Price of Defense.

Matching Nash Equilibria. Introduced in [9], a *Matching Nash equilibrium* satisfies several (necessary) *covering* properties of Nash equilibria and two additional properties (for example, the support of the vertex players is an *Independent Set*); in addition, all vertex players use a common distribution, and each player uses a uniform distribution on its support.

– We provide a new characterization of graphs admitting Matching Nash equilibria (Theorem 5). Such graphs have their *Independence Number* equal to their *Edge Covering Number.* The characterization improves an earlier one from [9]. The characterization benefits from an improved understanding of structural properties of Matching Nash equilibria.
– We translate the characterization into a polynomial time algorithm to (decide the existence of and) compute a Matching Nash equilibrium (Theorem 6). This relies on obtaining a polynomial time algorithm for the (new) graph-theoretic problem of deciding, given a graph $G(V, E)$, whether its *Independence Number* $\alpha(G)$ and *Edge Covering Number* $\beta'(G)$ are equal, and yielding, if so, a Maximum Independent Set for the graph (Proposition 1).
– We prove that the Price of Defense for them is $\alpha(G)$ (Theorem 7).

Perfect Matching Nash Equilibria. A *Perfect Matching Nash equilibrium* is a Perfect Matching one where the support of the edge player is a *Perfect Matching.*

– We provide a characterization of graphs admitting a Perfect Matching Nash equilibria (Theorem 8). Such graphs have a Perfect Matching and their Independence Number equals $\frac{|V|}{2}$.
– We translate the characterization into a polynomial time algorithm to (decide the existence of and) compute a Perfect Matching Nash equilibrium (Theorem 9). This relies on obtaining a polynomial time algorithm for the (new) graph-theoretic problem of deciding, given a graph with a Perfect Matching, whether its Independence Number equals $\frac{|V|}{2}$, and yielding, if so, a Maximum Independent Set for the graph (Proposition 2).
– We prove that the Price of Defense for them is $\frac{|V|}{2}$ (Theorem 10).

The relation between the Prices of Defense for Perfect Matching and Matching Nash equilibria is the relation between $\frac{|V|}{2}$ and $\alpha(G)$. For graphs that have both Matching and Perfect Matching Nash equilibria, Theorem 8 implies that $\alpha(G) = \frac{|V|}{2}$ and the two Prices of Defense coincide (as do the two classes of equilibria). Consider a graph that has a Matching Nash equilibrium but not a Perfect Matching one. By the characterization of Matching Nash equilibria in [9, Theorem 3], $\alpha(G) \geq \frac{|V|}{2}$ (else, there could not be enough vertices inside an Independent Set to which vertices outside have to be matched). Thus, the Price of Defense for Perfect Matching Nash equilibria may not exceed that for Matching Nash equilibria.

Defender Uniform Nash Equilibria. In a *Defender Uniform Nash equilibrium*, the defender chooses each edge in its support with uniform probability. Such equilibria are inspired by the recent *Uniform Nash equilibria* introduced by Bonifaci *et. al.* [1,2] for (classes of) bimatrix games. Bonifaci *et al.* [1,2] proved that deciding the existence of Uniform Nash equilibria is $\mathcal{NP}$-complete.

– We provide a characterization of graphs admitting Defender Uniform Nash equilibria (Theorem 11). The characterization involves *Regular Subgraphs* and Independent Sets. (Remarkably, Regular Subgraphs were also encountered in the work of Bonifaci *et al.* [1,2].)
– We prove that deciding the existence of a Defender Uniform Nash equilibrium is $\mathcal{NP}$-complete (Theorem 12). This employs a reduction from the UNDIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST SIX problem, which is proved $\mathcal{NP}$-complete (Theorem 3). Our $\mathcal{NP}$-completeness result strengthens the corresponding $\mathcal{NP}$-completeness results of Bonifaci *et al.* [1,2] since it applies to a *specific* game.
– We prove that the Price of Defense for them is $(\pi + 1) \cdot |V|$, for some $0 \leq \pi \leq \frac{1}{2}$. We argue that this value is always between those for Perfect Matching and Matching Nash equilibria.

Compared with Matching Nash equilibria, Defender Uniform Nash equilibria provide a sometimes better Price of Defense, but they are hard to compute (unless $\mathcal{P} = \mathcal{NP}$).

In an *Attacker Symmetric Uniform Nash equilibrium*, there is a common support for attackers and each attacker uses a uniform probability distribution.

– We provide a characterization of graphs admitting Attacker Symmetric Uniform Nash equilibria (Theorem 11).
– We translate the characterization into a polynomial time algorithm to (decide the existence and) compute an Attacker Symmetric Uniform Nash equilibrium (Theorem 15). This is in contrast to the $\mathcal{NP}$-completeness for the relative class of Defender Uniform Nash equilibria.
– We prove that the Price of Defense here is either $\frac{|V|}{2}$ or $\alpha(G)$ (Theorem 16).

2 Background, Definitions and Preliminaries

Graph Theory. Throughout, we consider an undirected graph $G = \langle V, E \rangle$ with no isolated vertices. For a vertex set $U \subseteq V$, denote $G(U)$ the subgraph of G induced by U, and $\mathsf{Edges}_G(U) = \{(u, v) \in E : \; u \in U \text{ and } v \in U\}$. For an edge set $F \subseteq E$, denote $G(F)$ the subgraph of G induced by F. For a vertex set $U \subseteq V$, denote $\mathsf{Neigh}_G(U) = \{u \notin U : (u, v) \in E \text{ for some vertex } v \in U\}$. For a vertex $v \in V$, denote $d_G(v)$ the degree of vertex v in G.

A vertex set $IS \subseteq V$ is an *Independent Set* if for all pairs of vertices $u, v \in IS$, $(u, v) \notin E$. A *Maximum Independent Set* is one that has maximum size; denote $\alpha(G)$ its size. A *Vertex Cover* is a vertex set $VC \subseteq V$ such that for each edge $(u, v) \in E$ either $u \in VC$ or $v \in VC$. A *Minimum Vertex Cover* is one that has minimum size; denote $\beta(G)$ its size. An *Edge Cover* is an edge set $EC \subseteq E$ such that for every vertex $v \in V$, there is an edge $(v, u) \in EC$. A *Minimum Edge Cover* is one that has maximum size; denote $\beta'(G)$ its size.

A *Matching* is a set $M \subseteq E$ non-incident edges. A *Maximum Matching* is one that has maximum size; $\alpha'(G)$ denotes its size. The currently fastest algorithm to compute a Maximum Matching runs in time $O\left(\sqrt{|V|}|E| \cdot \log_{|V|} \frac{|V|^2}{|E|}\right)$ [6]. It is known that a Minimum Edge Cover can be computed in polynomial time via computing a Maximum Matching. (See, e.g., [15, page 115].) A *Perfect Matching* is a Matching that is also an Edge Cover. For a graph G, $\alpha(G) + \beta(G) = |V|$, while also $\alpha'(G) + \beta'(G) = |V|$ (*Gallai's Theorem*). Also, $\alpha'(G) \leq \beta(G)$. Hence, $\alpha(G) \leq \beta'(G)$.

Fix a set $U \subseteq V$. G is a U*-Expander* graph (and the set U is an *Expander*) if for each set $U' \subseteq U$, $|U'| \leq |\mathsf{Neigh}_G(U') \cap (V \backslash U)|$. An *Expanding Independent Set* [9] is an Independent Set IS such that $V \backslash IS$ is an Expander.

Game Theory. Consider a *strategic game* $\Pi(G) = \langle \mathcal{N}, \{S_i\}_{i \in \mathcal{N}}, \{\mathsf{IP}\}_{i \in \mathcal{N}} \rangle$:

– The set of *players* is $\mathcal{N} = \mathcal{N}_{vp} \cup \mathcal{N}_{ep}$, where $\mathcal{N}_{vp}$ has ν *vertex* players vp_i, called *attackers*, $1 \leq i \leq \nu$ and $\mathcal{N}_{ep}$ has *edge* player ep, called *defender*.
– The *strategy set* S_i of vertex player vp_i is V, and the *strategy set* S_{ep} of the edge player ep is E. So, the *strategy set* $\mathcal{S}$ of the game is $\mathcal{S} = \left(\underset{i \,\in\, N_{vp}}{\times} S_i \right) \times S_{ep} = V^{\nu} \times E$.
– Fix any *profile* $\mathbf{s} = \langle s_1, \ldots, s_{\nu}, s_{ep} \rangle \in \mathcal{S}$, also called a *pure profile*.
  - The *Individual Profit* of vertex player vp_i is a function $\mathsf{IP}_{\mathbf{s}}(i) : \mathcal{S} \rightarrow \{0, 1\}$ such that $\mathsf{IP}_{\mathbf{s}}(i) = \begin{cases} 0, & s_i \in s_{ep} \\ 1, & s_i \notin s_{ep} \end{cases}$; intuitively, the vertex player vp_i receives 1 if it is not caught by the edge player, and 0 otherwise.
  - The *Individual Profit* of the edge player ep is a function $\mathsf{IP}_{\mathbf{s}}(ep) : \mathcal{S} \rightarrow \mathbb{N}$ such that $\mathsf{IP}_{\mathbf{s}}(ep) = |\{i : s_i \in s_{ep}\}|$; intuitively, the edge player ep receives the number of vertex players it catches.

A *mixed strategy* for player $i \in \mathcal{N}$ is a probability distribution over S_i; thus, a mixed strategy for a vertex player (resp., edge player) is a probability distribution over vertices (resp., edges) of G. A *profile* $\mathbf{s} = \langle s_1, \ldots, s_{\nu}, s_{ep} \rangle$ is a collection of mixed strategies; $s_i(v)$ is the probability that vertex player vp_i chooses vertex v and $s_{ep}(e)$ is the probability that the edge player ep chooses edge e.

The *support* $\mathsf{Support}_{\mathbf{s}}(i)$ of player $i \in \mathcal{N}$ in the profile $\mathbf{s}$ is the set of pure strategies to which i assigns a strictly positive probability. Denote $\mathsf{Support}_{\mathbf{s}}(vp) = \bigcup_{i \in \mathcal{N}_{vp}} \mathsf{Support}_{\mathbf{s}}(i)$ and $\mathsf{Edges}_{\mathbf{s}}(v) = \{(u, v) \in E : (u, v) \in \mathsf{Support}_{\mathbf{s}}(ep)\}$.

A profile $\mathbf{s}$ is *Uniform* if each player uses a uniform distribution on its support. A profile $\mathbf{s}$ is *Attacker Symmetric* if, for all vertex players vp_i, $vp_k \in \mathcal{N}_{vp}$, $s_i(v) = s_k(v)$, for each $v \in V$. An *Attacker Symmetric Uniform* profile is an Attacker Symmetric profile where each attacker uses a uniform distribution on the common support. A profile is *Defender Uniform* if the edge player uses a uniform distribution on its support.

For a vertex $v \in V$, the probability that the edge player ep chooses an edge containing the vertex v is denoted as $P_{\mathbf{s}}(\mathsf{Hit}(v))$. For a vertex $v \in V$, denote as

$\mathsf{VP}_{\mathbf{s}}(v)$ the expected number of vertex players choosing vertex v. For each edge $e = (u, v) \in E$, $\mathsf{VP}_{\mathbf{s}}(e)$ is the expected number of vertex players choosing either u or v.

A profile $\mathbf{s}$ induces an *Expected Individual Profit* $\mathsf{IP}_{\mathbf{s}}(i)$ for each player $i \in \mathcal{N}$, which is the expectation according to $\mathbf{s}$ of its Individual Profit. The profile $\mathbf{s}$ is a (mixed) *Nash equilibrium* [11,12] (abbreviated as *NE*) if for each player $i \in \mathcal{N}$, it maximizes $\mathsf{IP}_{\mathbf{s}}(i)$ over all profiles that differ from $\mathbf{s}$ only with respect to the mixed strategy of player i. We use a characterization of NE from [9]:

Theorem 1 ([9]). *A profile* $\mathbf{s}$ *is a Nash equilibrium if and only if* (1) *for each vertex* $v \in \mathsf{Support}_{\mathbf{s}}(vp)$, $P_{\mathbf{s}}(\mathsf{Hit}(v)) = \min_{v' \in V} P_{\mathbf{s}}(\mathsf{Hit}(v'))$ *and* (2) *for each edge* $e \in \mathsf{Sup-\ port}_{\mathbf{s}}(ep)$, $\mathsf{VP}_{\mathbf{s}}(e) = \max_{e' \in E} \mathsf{VP}_{\mathbf{s}}(e')$.

A *Covering profile* is a profile $\mathbf{s}$ such that (1) $\mathsf{Support}_{\mathbf{s}}(ep)$ is an Edge Cover and (2) $\mathsf{Support}_{\mathbf{s}}(vp)$ is a Vertex Cover of the graph $G(\mathsf{Support}_{\mathbf{s}}(ep))$. It is shown in [9] that a Nash equilibrium $\mathbf{s}$ is a Covering profile. (It is also shown in [9] that a Covering profile is not necessarily a Nash equilibrium.) An *Independent Covering profile* [9] is a Uniform, Attacker Symmetric, Covering profile $\mathbf{s}$ such that (1) $\mathsf{Support}_{\mathbf{s}}(vp)$ is an Independent Set of G and (2) each vertex in $\mathsf{Support}_{\mathbf{s}}(vp)$ is incident to exactly one edge in $\mathsf{Support}_{\mathbf{s}}(ep)$. Clearly, by the fact that $\mathbf{s}$ is a Covering profile and Condition (2), it follows that for an Independent Covering profile $\mathbf{s}$, $|\mathsf{Support}_{\mathbf{s}}(vp)| = |\mathsf{Support}_{\mathbf{s}}(ep)|$. It is finally shown in [9] that for an Independent Covering profile $\mathbf{s}$, there is a Matching $M \subseteq \mathsf{Support}_{\mathbf{s}}(ep)$ that matches each vertex in $V \backslash \mathsf{Support}_{\mathbf{s}}(vp)$ to some vertex in $\mathsf{Support}_{\mathbf{s}}(vp)$; note that $|M| = |V \backslash \mathsf{Support}_{\mathbf{s}}(vp)|$. The same work shows that an Independent Covering profile is a NE, called a *Matching NE* [9]. A graph-theoretic characterization of Matching NE is provided there:

Theorem 2 ([9]). *A graph G admits a Matching NE if and only if G has an Expanding Independent Set.*

We study algorithmic problems of existence and computation of various classes of Nash equilibria for the considered game. CLASS NE EXISTENCE asks whether $\Pi(G)$ admits a CLASS NE; FIND CLASS NE search for a CLASS NE of $\Pi(G)$, assuming that such exists. Variable CLASS takes values GENERAL, MATCHING, PERFECT MATCHING, DEFENDER UNIFORM and ATTACKER SYMMETRIC UNIFORM; it determines the classes of general, Matching, Perfect Matching, Defender Uniform Attacker Symmetric Uniform Nash equilibria, respectively. We note that for all values of CLASS, membership of a profile in CLASS can be verified in polynomial time. Since a Nash equilibrium can be verified in polynomial time (by Theorem 1), it follows that CLASS NE EXISTENCE $\in \mathcal{NP}$.

The Price of Defense (abbreviated as PoD_G) is the *worst-case* ratio, over all Nash equilibria $\mathbf{s}$, of $\frac{\nu}{\mathsf{IP}_{\mathbf{s}}(ep)}$.

3 Some Problems from Graph Theory

For our negative results, we will use two $\mathcal{NP}$-complete graph-theoretic problems, stated here in the style of Garey and Johnson [4]:

DIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST THREE
INPUT: A directed graph $G_d(V_d, E_d)$.
QUESTION: Can the vertex set V_d be partitioned into disjoint sets $V_1, \cdots V_k$, for some k, such that each V_i contains at least three vertices and induces a Hamiltonian subgraph?

The problem DIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST THREE is $\mathcal{NP}$-complete [14]. To the best of our knowledge, the following problem is new:

UNDIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST SIX
INPUT: An undirected graph $G(V, E)$.
QUESTION: Can the vertex set V be partitioned into disjoint sets $V_1, \cdots V_k$, for some k, such that each V_i contains at least six vertices and induces a Hamiltonian subgraph?

There is a variant of this problem, UNDIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST SIX, which is cited in [4, GT13] as $\mathcal{NP}$-complete. We strengthen this result by proving:

Theorem 3. UNDIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST SIX *is $\mathcal{NP}$-complete.*

Sketch of Proof. UNDIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST SIX $\in \mathcal{NP}$: one can guess a partition of V into disjoint sets $V_1, \cdots V_k$, each of size at least six, together with a vertex sequence for each set V_i, and verify in polynomial time that the vertex sequence for each set V_i is a Hamiltonian circuit for the subgraph induced by the set. To prove $\mathcal{NP}$-hardness, we reduce from DIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST THREE. □

For our positive results, we will consider two new graph-theoretic problems:

MIS EQUAL MINIMUM EDGE COVER
INSTANCE: A graph $G(V, E)$.
OUTPUT: A Maximum Independent Set of size $\beta'(G)$ if $\alpha(G) = \beta'(G)$, or NO if $\alpha(G) < \beta'(G)$.

MIS EQUAL HALF ORDER
INSTANCE: A graph $G(V, E)$.
OUTPUT: A Maximum Independent Set of size $\frac{|V|}{2}$ if $\alpha(G) = \frac{|V|}{2}$, or NO if $\alpha(G) \neq \frac{|V|}{2}$.

For these two new problems, we use reductions to 2SAT to prove:

Proposition 1. MIS EQUAL MINIMUM EDGE COVER $\in \mathcal{P}$

Sketch of Proof. Compute a Minimum Edge Cover EC. (Recall that EC consists of vertex-disjoint star graphs.) Use EC to construct a 2SAT instance ϕ with variable set V as follows:

(1) For each edge $(u, v) \in E$, add the clause $(\bar{u} \vee \bar{v})$ to ϕ.
(2) For each single-edge star graph $(u, v) \in EC$, add the clause $(u \vee v)$ to ϕ.
(3) For each multiple-edge star graph of EC with *center* vertex u, add the clause $(\bar{u} \vee \bar{u})$ to ϕ.

We prove that G has an Independent Set of size $|EC|$ (hence, $\alpha(G) = \beta'(G)$) if and only if ϕ is satisfiable; when ϕ is satisfiable, the set $\{u \mid \chi(u) = 1\}$ is such a Maximum Independent Set. □

Proposition 2. MIS EQUAL HALF ORDER $\in \mathcal{P}$, *when restricted to the class of graphs having a Perfect Matching.*

Sketch of Proof. Compute a Perfect Matching M of G. Use M to construct a 2SAT instance ϕ with variable set V as follows:

(1) For each edge $(u, v) \in E$, add the clause $(\bar{u} \vee \bar{v})$ to ϕ.
(2) For each edge $(u, v) \in M$, add the clause $(u \vee v)$ to ϕ.

We prove that G has $\alpha(G) = \frac{|V|}{2}$ if and only if ϕ is satisfiable; if ϕ is satisfiable, the set $\{u \mid \chi(u) = 1\}$ is such a Maximum Independent Set of size $\frac{|V|}{2}$. □

4 General Nash Equilibria

Denote as $\hat{\Pi}(G)$ the Two-Players special case of $\Pi(G)$ with $\nu = 1$. Consider a NE $\hat{\mathbf{s}}$ of $\hat{\Pi}(G)$. Construct from $\hat{\mathbf{s}}$ an Attacker Symmetric profile $\mathbf{s}$ for $\Pi(G)$, where for each vertex player vp_i, for each vertex $v \in V$, $s_i(v) = \hat{s}_{vp}(v)$, where vp denotes the vertex player of $\hat{\Pi}(G)$; for the edge player ep, for each edge $e \in E$, $s_{ep}(e) = \hat{s}_{ep}(e)$. We prove that $\mathbf{s}$ satisfies the characterization of NE in Theorem 1; so, $\mathbf{s}$ is a NE for $\Pi(G)$ and can be computed from $\hat{\mathbf{s}}$ in polynomial time. We prove that the Two-Players game $\hat{\Pi}(G)$ is *Constant-Sum*:

$$\begin{aligned}
\mathsf{IP}_{\hat{\mathbf{s}}}(vp) + \mathsf{IP}_{\hat{\mathbf{s}}}(ep) &= \sum_{v \in V} \hat{s}_{vp}(v) \left(1 - \sum_{e \in \mathsf{Edges}_{\mathbf{s}}(v)} \hat{s}_{ep}(e)\right) + \sum_{(u,v)=e\, \in E} \hat{s}_{ep}(e)\hat{s}_{vp}(e) \\
&= \sum_{v \in V} \hat{s}_{vp}(v) - \sum_{v \in V} \hat{s}_{vp}(v) \sum_{e \in \mathsf{Edges}_{\mathbf{s}}(v)} \hat{s}_{ep}(e) + \sum_{(u,v)=e\, \in E} \hat{s}_{ep}(e)\hat{s}_{vp}(e) \\
&= 1 - \sum_{(u,v)=e\, \in E} \hat{s}_{ep}(e)\left(\hat{s}_{vp}(e)\right) + \sum_{(u,v)=e\, \in E} \hat{s}_{ep}(e)\hat{s}_{vp}(e) \\
&= 1.
\end{aligned}$$

Since a Nash equilibrium for a Two-Players, Constant-Sum game can be computed in polynomial time via reduction to Linear Programming [13], we have:

Theorem 4. FIND GENERAL NE $\in \mathcal{P}$

5 Matching Nash Equilibria

We first prove some graph-theoretic properties of Matching Nash equilibria.

Lemma 1. *In a Matching NE* $\mathbf{s}$*,* $\mathsf{Support}_{\mathbf{s}}(vp)$ *is a Maximum Independent Set.*

Lemma 2. *In a Matching NE* $\mathbf{s}$*,* $\mathsf{Support}_{\mathbf{s}}(ep)$ *is a Minimum Edge Cover.*

Theorem 5. *The graph G admits a Matching NE if and only if $\alpha(G) = \beta'(G)$.*

Sketch of Proof. Assume first that $\alpha(G) = \beta'(G)$. Let IS and EC be a Maximum Independent Set and a Minimum Edge Cover, respectively. So, $|IS| = |EC|$. Consider a Uniform, Attacker Symmetric profile $\mathbf{s}$ with $\mathsf{Support}_{\mathbf{s}}(vp) = IS$ and $\mathsf{Support}_{\mathbf{s}}(ep) = EC$. Thus, $|\mathsf{Support}_{\mathbf{s}}(vp)| = |\mathsf{Support}_{\mathbf{s}}(ep)|$. By construction, $\mathbf{s}$ is Uniform and Attacker Symmetric profile, $\mathsf{Support}_{\mathbf{s}}(ep)$ is an Edge Cover of G, and $\mathsf{Support}_{\mathbf{s}}(vp)$ is an Independent Set of G. So, there remains to show Condition (2) in the definition of a Covering profile and additional Condition (2) in the definition of an Independent Covering profile. Since EC is a *Minimum* Edge Cover, it is a union of disjoint star graphs. Since $|\mathsf{Support}_{\mathbf{s}}(vp)| = |\mathsf{Support}_{\mathbf{s}}(ep)|$ and $\mathsf{Support}_{\mathbf{s}}(vp)$ is an Independent Set, it follows that $\mathsf{Support}_{\mathbf{s}}(vp)$ consists of *all* terminal vertices of the star graphs. This implies that both *(i)* $\mathsf{Support}_{\mathbf{s}}(vp)$ is a Vertex Cover of the graph $G(\mathsf{Support}_{\mathbf{s}}(ep))$ (Condition (2)) and *(ii)* each vertex in $\mathsf{Support}_{\mathbf{s}}(vp)$ is incident to exactly one edge of $\mathsf{Support}_{\mathbf{s}}(ep)$ (additional Condition (2)). Hence, $\mathbf{s}$ is an Independent Covering profile. Since an Independent Covering profile is a NE, the claim follows. Assume now that G admits a Matching NE $\mathbf{s}$. By Lemma 1, $\mathsf{Support}_{\mathbf{s}}(vp)$ is a Maximum Independent Set, so that $|\mathsf{Support}_{\mathbf{s}}(vp)| = \alpha(G)$. By Lemma 2, $\mathsf{Support}_{\mathbf{s}}(ep)$ is a Minimum Edge Cover, so that $|\mathsf{Support}_{\mathbf{s}}(ep)| = \beta'(G)$. Since $\mathbf{s}$ is a Matching NE, $|\mathsf{Support}_{\mathbf{s}}(vp)| = |\mathsf{Support}_{\mathbf{s}}(ep)|$. So, $\alpha(G) = \beta'(G)$. □

The constructive parts of the sufficiency proofs of Proposition 1 and Theorem 5 yield a polynomial time algorithm MatchingNE to compute a Matching NE:

```
Algorithm MatchingNE
INPUT: A graph $G(V, E)$.
OUTPUT: The supports in a Matching NE $\mathbf{s}$ for $G$, or NO if such does not exist.

 1. Compute a Minimum Edge Cover $EC$ of $G$.
 2. Construct an instance $\phi$ of 2SAT as follows:
      – For each edge $(u, v) \in E$, add the clause $(\bar{u} \vee \bar{v})$ to $\phi$.
      – For each single-edge star graph $(u, v) \in EC$, add the clause $(u \vee v)$ to $\phi$.
      – For each multiple-edge star graph of $EC$ with center vertex $u$, add the clause
        $(\bar{u} \vee \bar{u})$ to $\phi$.
 3. Compute a satisfying assignment $\chi$ of $\phi$, or output NO if such does not exist.
 4. Set $IS = \{u \mid \chi(u) = 1\}$.
 5. Set $\mathsf{Support}_{\mathbf{s}}(ep) := EC$ and $\mathsf{Support}_{\mathbf{s}}(vp) := IS$.
```

Theorem 6. *Algorithm* MatchingNE *solves* FIND MATCHING NE *in time* $O\left(\sqrt{|V|}|E| \cdot \log_{|V|} \frac{|V|^2}{|E|}\right)$.

By Lemma 1 and the definition of a Covering profile, we can easily show:

Theorem 7. *For Matching NE,* $\mathsf{PoD}_G = \alpha(G)$.

6 Perfect Matching Nash Equilibria

A *Perfect Matching NE* is a Matching NE $\mathbf{s}$ such that $\mathsf{Support}_{\mathbf{s}}(ep)$ is a Perfect Matching. To obtain the characterization, we prove:

Lemma 3. *For a Perfect Matching NE* $\mathbf{s}$, $|\mathsf{Support}_{\mathbf{s}}(vp)| = \frac{|V|}{2}$.

Theorem 8. *A graph G admits a Perfect Matching NE if and only if G has a Perfect Matching and $\alpha(G) = \frac{|V|}{2}$.*

Sketch of Proof. Assume first that G has a Perfect Matching M and $\alpha(G) = \frac{|V|}{2}$. Consider a Maximum Independent Set IS of G. Define a Uniform, Attacker Symmetric profile $\mathbf{s}$ with $\mathsf{Support}_{\mathbf{s}}(ep) := M$, $\mathsf{Support}_{\mathbf{s}}(vp) := IS$. By the choice of $\mathsf{Support}_{\mathbf{s}}(ep)$, we only need to prove that $\mathbf{s}$ is an Independent Covering profile. Since a Perfect Matching is a Minimum Edge Cover, this reduces to the corresponding proof of Theorem 5 (where $\mathsf{Support}_{\mathbf{s}}(ep)$ was chosen as a Minimum Edge Cover). Assume now that G admits a Perfect Matching NE $\mathbf{s}$. Then, $\mathsf{Support}_{\mathbf{s}}(ep)$ is a Perfect Matching of G. Since a Perfect Matching NE is a special case of a Matching NE, Lemma 1 applies to yield that $\mathsf{Support}_{\mathbf{s}}(vp) = \alpha(G)$. By Lemma 3, $\mathsf{Support}_{\mathbf{s}}(vp) = \frac{|V|}{2}$. So, $\alpha(G) = \frac{|V|}{2}$, as needed. □

The constructive parts of the sufficiency proofs of Proposition 2 and Theorem 8 yield a polynomial time algorithm to compute a Perfect Matching NE:

```
Algorithm PerfectMatchingNE
INPUT: A graph $G(V, E)$.
OUTPUT: The supports in a Perfect Matching NE $\mathbf{s}$, or NO if such does not exist.

1. Compute a Perfect Matching $M$ of $G$, or output NO if such does nots exist.
2. Construct an instance $\phi$ of 2SAT as follows:
     – For each edge $(u, v) \in E$, add the clause $(\bar{u} \vee \bar{v})$ to $\phi$.
     – For each edge $(u, v) \in M$, add the clause $(u \vee v)$ to $\phi$.
3. Compute a satisfying assignment $\chi$ of $\phi$, or output NO if such does not exist.
4. Set $IS = \{u \mid \chi(u) = 1\}$.
5. Set $\mathsf{Support}_{\mathbf{s}}(ep) := M$ and $\mathsf{Support}_{\mathbf{s}}(vp) := IS$.
```

Theorem 9. *Algorithm* PerfectMatchingNE *solves* FIND PERFECT MATCHING NE *in time* $O\left(\sqrt{|V|}|E| \cdot \log_{|V|} \frac{|V|^2}{|E|}\right)$.

Observe that a Perfect Matching NE is a Matching NE for which, by Theorem 8, $\alpha(G) = \frac{|V|}{2}$. Hence, Theorem 7 implies:

Theorem 10. *For Perfect Matching NE,* $\mathsf{PoD}_G = \frac{|V|}{2}$.

7 Defender Uniform Nash Equilibria

A *Defender Uniform NE* is a Defender Uniform profile that is a NE. We prove a characterization of graphs admitting Defender Uniform NE:

Theorem 11. *A graph G admits a Defender Uniform NE if and only if there are non-empty sets $V' \subseteq V$ and $E' \subseteq E$ and an integer $r \geq 1$ such that:*

(1/a) *For each $v \in V'$, $d_{G(E')}(v) = r$.*
(1/b) *For each $v \in V \backslash V'$, $d_{G(E')}(v) \geq r$.*
(2) *V' can be partitioned into two disjoint sets V'_i and V'_r such that:*
  (2/a) *For each $v \in V'_i$, for any $u \in \mathsf{Neigh}_G(v)$, it holds that $u \notin V'$.*
  (2/b) *The graph $\langle V'_r, \mathsf{Edges}_G(V'_r) \cap E' \rangle$ is an r-regular graph.*
  (2/c) *The graph $\langle V'_i \cup (V\backslash V'), \mathsf{Edges}_G(V'_i \cup (V\backslash V')) \cap E' \rangle$ is a $(V'_i, V \backslash V')$ -bipartite graph.*

Theorem 12. DEFENDER UNIFORM NE EXISTENCE *is $\mathcal{NP}$-complete.*

Sketch of Proof. Recall that DEFENDER UNIFORM NE EXISTENCE $\in \mathcal{NP}$. To prove $\mathcal{NP}$-hardness, we reduce from the $\mathcal{NP}$-complete UNDIRECTED PARTITION INTO HAMILTONIAN SUBGRAPHS OF SIZE AT LEAST FOUR. □

By the characterization of Defender Uniform NE (Theorem 11, we prove:

Theorem 13. *In a Defender Uniform Nash equilibrium, $\mathsf{PoD}_G = (\pi + 1) \cdot |V|$, for some $0 \leq \pi \leq \frac{1}{2}$.*

The worst case (maximum value) of PoD_G occurs when $\mathsf{Support}_{\mathbf{s}}(vp)$ is the *maximum* possible and V'_r is empty. Then, the Defender Uniform NE $\mathbf{s}$ is actually a Matching NE and, by Proposition 7, $\mathsf{PoD}_G = \alpha(G)$. For the best case (minimum value), recall that by Condition (2/b) of the characterization of Defender Uniform NE, the graph $\langle V'_i \cup (V\backslash V'), \mathsf{Edges}_G(V'_i \cup (V\backslash V')) \cap E' \rangle$ is a $(V\backslash V')$-Expander graph. Thus, $|(V\backslash V')| \leq |V'_i|$. Since $|V| = |V'_r| + |V'_i| + |(V\backslash V')|$, it follows that $|V'_r| + 2|V'_i| \geq |V|$. Hence, $|V'_i| + \frac{|V'_r|}{2} \geq \frac{|V| - |V'_r|}{2} = \frac{|V|}{2}$. So, Defender Uniform NE fall between Perfect Matching and Matching NE (with respect to PoD_G), for the case where $\frac{|V|}{2} \leq \alpha(G)$.

8 Attacker Symmetric Uniform Nash Equilibria

An *Attacker Symmetric Uniform NE* is an Attacker Symmetric Uniform profile that is a NE. We prove a characterization of graphs admitting Attacker Symmetric Uniform NE:

Theorem 14. *A graph G admits an Attacker Symmetric Uniform Nash equilibrium if and only if:*

(1) *There is a probability distribution $p : E \rightarrow [0, 1]$ such that:*
(1/a) $\sum_{e \in \mathsf{Edges}_G(v)} p(e) = \sum_{e' \in \mathsf{Edges}_G(v')} p(e')$, $\forall\, v, v' \in V$ *and*

(1/b) $\sum_{e \in \mathsf{Edges}_G(v)} p(e) > 0 \ \forall \ v \in V$, OR
(2) $\alpha(G) = \beta'(G)$.

Theorem 15. FIND ATTACKER SYMMETRIC UNIFORM NE $\in \mathcal{P}$

Using Theorem 14 and Theorem 7, we finally prove:

Theorem 16. *In an Attacker Symmetric Uniform NE,* PoD_G *is* $\frac{|V|}{2}$ *or* $\alpha(G)$.

References

1. V. Bonifaci, U. Di Iorio and L. Laura, "On the Complexity of Uniformly Mixed Nash Equilibria and Related Regular Subgraph Problems", *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory*, pp. 197–208, Vol. 3623, LNCS, 2005.
2. V. Bonifaci, U. Di Iorio and L. Laura, "New Results on the Complexity of Uniformly Mixed Nash Equilibria", *Proceedings of the First International Workshop on Internet and Network Economics*, pp. 1023–1032, Vol. 3828, LNCS, 2005.
3. E. R. Cheswick and S. M. Bellovin, *Firewalls and Internet Security*, Addison-Wesley, 1994.
4. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., 1979.
5. M. Gelastou, M. Mavronicolas, V. Papadopoulou, A. Philippou and P. Spirakis, "The Power of the Defender", *Proceedings of the 2nd International Workshop on Incentive-Based Computing*, 2006, to appear.
6. A. V. Goldberg and A. V. Karzanov, "Maximum Skew-Symmetric Flows", *Proceedings of the 3rd Annual European Symposium on Algorithms*, pp. 155–170, Vol. 979, LNCS, 1995.
7. E. Koutsoupias and C. H. Papadimitriou, "Worst-case Equilibria", *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pp. 404–413, Vol. 1563, LNCS, 1999.
8. T. Markham and C. Payne, "Security at the Network Edge: A Distributed Firewall Architecture", *Proceedings of the 2nd DARPA Information Survivability Conference and Exposition*, Vol. 1, pp. 279-286, 2001.
9. M. Mavronicolas, V. G. Papadopoulou, A. Philippou and P. G. Spirakis. "A Network Game with Attacker and Protector Entities", *Proceedings of the 16th Annual International Symposium on Algorithms and Computation*, pp. 288–297, Vol. 3827, LNCS, 2005.
10. M. Mavronicolas, V. G. Papadopoulou, A. Philippou and P. G. Spirakis, "A Graph-Theoretic Network Security Game", *Proceedings of the First International Workshop on Internet and Network Economics*, pp. 969–978, Vol. 3828, LNCS, 2005.
11. J. F. Nash, "Equilibrium Points in N-Person Games", *Proceedings of National Acanemy of Sciences of the United States of America*, pp. 48–49, Vol. 36, 1950.
12. J. F. Nash, "Non-Cooperative Games", *Annals of Mathematics*, Vol. 54, No. 2, pp. 286–295, 1951.
13. J. von Neumann, "Zur Theorie der Gesellschaftsspiele", *Mathematische Annalen*, Vol. 100, pp. 295–320, 1928.
14. L. G. Valiant, "The Complexity of Computing the Permanent", *Theoretical Computer Science*, Vol. 8, No. 2, pp. 189–201, 1979.
15. D. B. West, *Introduction to Graph Theory*, Prentice Hall, Second edition, 2001.

The Data Complexity of MDatalog in Basic Modal Logics

Linh Anh Nguyen

Institute of Informatics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl

Abstract. We study the data complexity of the modal query language MDatalog and its extension eMDatalog in basic modal logics. MDatalog is a modal extension of Datalog, while eMDatalog is the general modal Horn fragment with the allowedness condition. As the main results, we prove that the data complexity of MDatalog and eMDatalog in $K4$, $KD4$, and $S4$ is PSPACE-complete, in K is coNP-complete, and in KD, T, KB, KDB, and B is PTIME-complete.

1 Introduction

Modal logics can be used to reason about knowledge and belief. It is desirable to study modal extensions of deductive databases. First tries in this direction were done in our previous works [9,10] (modal Datalog defined in [5, Definition 23] is completely different, as it is formulated in classical logic and uses only unary or binary predicates). In [9], we extended Datalog for monomodal logics, giving two languages: MDatalog and eMDatalog. The first one is a natural extension of Datalog, while the second one is the general modal Horn fragment with a refined condition of allowedness. It was shown in [9] that MDatalog and eMDatalog have the same expressiveness in normal monomodal logics. In [10], we studied an extension of MDatalog for multimodal logics of belief and presented bottom-up computational methods for (multi)modal deductive databases.

It is well known that the data complexity of Datalog is complete in PTIME (see, e.g., [6]). In [10], we proved that the data complexity of MDatalog in some multimodal logics of belief which are extensions of $KD45$ is in PTIME. In [9], we gave sufficient conditions for MDatalog in 13 basic monomodal logics (except K and $K4$) to obtain PTIME complexity for computing queries. The complexity results of [9, Theorem 7.3] are not formulated in the way of the data complexity and the relation between them is not close.

In this paper, we study the data complexity of MDatalog and eMDatalog in all of the 15 basic monomodal logics, which are extensions of the logic K using any combination of axioms D, T, B, 4, and 5. The data complexity of MDatalog and eMDatalog is related to the complexity of the propositional modal Horn fragment. In [3], Fariñas del Cerro and Penttonen showed that the satisfiability problem of sets of modal Horn clauses in $S5$ is decidable in PTIME. In [1], Chen and Lin showed that the similar problem for a normal monomodal logic L being an extension of $K5$ (write $K5 \leq L$) is also decidable in PTIME. Chen and Lin also proved that for a normal modal logic L such that $K \leq L \leq S4$ or $K \leq L \leq B$, in particular, for $L \in \{K, KD, KB, KDB, B, K4, KD4, S4\}$, the problem is PSPACE-hard. They also made a comment that the problem is still PSPACE-hard for $S4$ even when the modal depth is restricted to 2. In [8], we showed that the complexity of the satisfiability problem of sets of modal Horn clauses with finitely bounded modal depth in KD, T, KB, KDB, and B is decidable in PTIME. These PTIME results can further be categorized as PTIME-complete. In [11], we showed that the satisfiability problem of sets of modal Horn clauses with modal depth bounded by $k \geq 2$ in the modal logics $K4$ and $KD4$ is PSPACE-complete, and in K is NP-complete.

In this paper we prove that, for L being any one of the 15 basic monomodal logics, the data complexity of MDatalog and eMDatalog in L is the same as the complexity of the unsatisfiability problem in L of the propositional modal Horn fragment with modal depth bounded by $k \geq 2$. This means that the data complexity of MDatalog and eMDatalog in $K4$, $KD4$, and $S4$ is PSPACE-complete, in K is coNP-complete, and in the remaining logics is PTIME-complete.

2 Preliminaries

2.1 Definitions for Modal Logics

The language for propositional modal logics extends the language of classical propositional logic with two modal operators $\Box$ and $\Diamond$.

A *Kripke frame* is a triple $\langle W, \tau, R\rangle$, where W is a nonempty set of possible worlds, $\tau \in W$ is the actual world, and R is a binary relation on W, called the accessibility relation. A *propositional Kripke model*, sometimes briefly called a model, is a tuple $\langle W, \tau, R, h\rangle$, where $\langle W, \tau, R\rangle$ is a Kripke frame and h is a function that maps each world of W to a set of primitive propositions. A *model graph* is a tuple $\langle W, \tau, R, H\rangle$, where $\langle W, \tau, R\rangle$ is a Kripke frame and H is a function that maps each world of W to a formula set.

Given a Kripke model $M = \langle W, \tau, R, h\rangle$ and a world $w \in W$, the *satisfaction relation* $\models$ is defined as usual for the classical connectives with two extra clauses for the modal operators as below:

$$\begin{array}{ll} M, w \models \Box\varphi & \text{iff } \forall v \in W.\ R(w,v) \text{ implies } M, v \models \varphi \\ M, w \models \Diamond\varphi & \text{iff } \exists v \in W.\ R(w,v) \text{ and } M, v \models \varphi. \end{array}$$

We say that φ is *satisfied at w in M* if $M, w \models \varphi$. We say that φ is *satisfied in M* and call M a *model of φ* if $M, \tau \models \varphi$.

If we consider all Kripke models, with no restrictions on R, we obtain a normal propositional modal logic with a standard Hilbert-style axiomatization K. Other normal propositional modal logics are obtained by adding to K certain axioms. The most popular axioms used for extending K are D, T, B, 4, and 5, which respectively correspond to seriality ($\forall x\, \exists y. R(x,y)$), reflexiveness, symmetry, transitiveness, and euclideaness ($\forall x\, \forall y\, \forall z. R(x,y) \wedge R(x,z) \rightarrow R(y,z)$) of the accessibility relation. The names of normal propositional modal logics often consist of K and the names of the added axioms, e.g. KDB is the logic which extends K with the axioms D and B. The special cases are T, B, $S4$, and $S5$, which stand for KT, KTB, $KT4$, and $KT5$, respectively.

We refer to the properties of the accessibility relation of a modal logic L as the *L-frame restrictions*. A Kripke model M is an *L-model* if the accessibility relation of M satisfies all L-frame restrictions. We say that φ is *L-satisfiable* if there exists an L-model of φ. We write $\Gamma \models_L \varphi$ to denote that φ is satisfied in every L-model of Γ.

The *modal depth* of a formula φ is the maximal nesting depth of modal operators occurring in φ; e.g., the modal depth of $p \wedge \Box(\Diamond q \vee \Diamond r)$, where p, q, r are primitive propositions, is 2.

The *length* of a formula φ is the total number of symbols occurring in φ. The *size* of a formula set is the sum of the lengths of its formulas.

The *size* of a propositional Kripke model $M = \langle W, \tau, R, h \rangle$ is the sum of the number of its worlds, the size of its accessibility relation, and the total number of primitive propositions from its worlds, i.e. $|W| + |R| + \Sigma_{w \in W}|h(w)|$.

In this work, we consider also first-order modal logics. We restrict to first-order modal logics with *fixed-domain* (i.e. all possible worlds in a first-order Kripke model have the same domain) and *rigid terms* (i.e. the semantics of terms does not depend on possible worlds). As usual for database models, we assume that the signature contains predicate symbols and constant symbols, but no function symbols. The definitions given for propositional modal logics can be shifted in a natural way for first-order modal logics.

We refer to [2,4] for further reading on modal logics.

2.2 The Modal Query Languages MDatalog and eMDatalog

An *MDatalog program clause* is a formula of the form

$$\forall x_1 \dots \forall x_k \, \Box^h (B_1 \wedge \dots \wedge B_l \to A)$$

where $\Box^h$ is a sequence of h operators $\Box$, $h \geq 0$, $l \geq 0$, and

- A, B_1, ..., B_l are of the form $\Box E$, $\Diamond E$, or E, with E being a classical atom (recall that a classical atom is a formula of the form $p(t_1, \dots, t_n)$),
- $x_1, \dots, x_k$ are all variables occurring in $(B_1 \wedge \dots \wedge B_l \to A)$,
- all variables of A occur also in $B_1 \wedge \dots \wedge B_l$.

The last condition in the above definition is usually called *allowedness* (or *range-restrictedness*). It implies that if $l = 0$ then $k = 0$ and A is a ground formula. We will write the above program clause in the form $\Box^h(A \leftarrow B_1, \dots, B_l)$. We will also write $\varphi \leftarrow \psi$ for $\psi \to \varphi$.

An *MDatalog program* is a set of MDatalog program clauses.

We now define an extension of MDatalog called *eMDatalog*, which allows more sophisticated program clauses.

A formula is *positive* if it does not contain $\neg$ and $\leftarrow$. A formula φ is called a *non-negative modal Horn formula* iff one of the following conditions holds:

– φ is a classical atom;
– φ is of the form $\psi \leftarrow \zeta$, where ψ is a non-negative modal Horn formula and ζ a positive formula without quantifiers such that if $\zeta_1 \vee \zeta_2$ is a subformula of ζ then ζ_1 and ζ_2 have the same variables[1];
– $\varphi = \Box\psi$, or $\varphi = \Diamond\psi$, or $\varphi = \psi \wedge \zeta$, where ψ and ζ are non-negative modal Horn formulas.

In the following, E denotes a classical atom and $Var(\varphi)$ denotes the set of variables of φ. Define the constraint $allowed(\varphi, V)$ for a non-negative modal Horn formula φ and a set of variables V recursively as follows:

$$\begin{aligned}
allowed(E, V) &\equiv (Var(E) \subseteq V)\\
allowed((\psi \leftarrow \zeta), V) &\equiv allowed(\psi, Var(\zeta) \cup V)\\
allowed(\psi \wedge \zeta, V) &\equiv allowed(\psi, V) \wedge allowed(\zeta, V)\\
allowed(\Box\psi, V) &\equiv allowed(\psi, V)\\
allowed(\Diamond\psi, V) &\equiv allowed(\psi, V)
\end{aligned}$$

An *eMDatalog program clause* is a formula of the form $\forall x_1 \ldots \forall x_k\, \varphi$, where φ is a non-negative modal Horn formula, $x_1, \ldots, x_k$ are all variables of φ, and the constraint $allowed(\varphi, \emptyset)$ is true. Observe that an eMDatalog program clause not containing $\leftarrow$ is a ground formula.

An *eMDatalog program* is a finite set of eMDatalog program clauses.

For $\overline{x}$ being a tuple of variables $(x_1, \ldots, x_k)$, we write $\exists\overline{x}\ \varphi(\overline{x})$ to denote the formula $\exists x_1 \ldots \exists x_k\, \varphi$ and assume that $x_1, \ldots, x_k$ are variables occurring in φ. In that case, for $\overline{c}$ being a tuple of constant symbols $(c_1, \ldots, c_k)$, we write $\varphi(\overline{c})$ to denote the formula obtained from φ by substituting each x_i by c_i, for $1 \leq i \leq k$.

A *query* to an MDatalog/eMDatalog program is a formula of the form $\exists\overline{x}\ \varphi(\overline{x})$, where $\overline{x}$ is a tuple of all variables of φ (which can be empty) and φ is a positive formula without quantifiers such that if $\zeta_1 \vee \zeta_2$ is a subformula of φ then ζ_1 and ζ_2 have the same variables[2].

Fix a first-order modal logic L. Given an MDatalog/eMDatalog program P and a query $\exists\overline{x}\ \varphi(\overline{x})$, the goal is to check whether $P \models_L \exists\overline{x}\ \varphi(\overline{x})$, and to find tuples $\overline{c}$ of constant symbols such that $P \models_L \varphi(\overline{c})$.

The following proposition states that MDatalog has the same expressiveness as eMDatalog. It was proved in [9] for the case without $\vee$ in non-negative modal Horn formulas. The proof for the extension with $\vee$ is straightforward.

Proposition 1. *For any eMDatalog program P, there exists an MDatalog program P' such that for any query $\exists\overline{x}\ \varphi(\overline{x})$ in the language of P, the answers w.r.t. P' are exactly the answers w.r.t. P. Moreover, P' can be computed from P in polynomial time and the modal depth of P' is equal to the modal depth of P.*

For example, the eMDatalog program $\{\Diamond(p(a) \wedge q(a))\}$ can be transformed to the MDatalog program $\{\Diamond r,\ \Box(p(a) \leftarrow r),\ \Box(q(a) \leftarrow r)\}$.

[1] This extends the corresponding definition given in [9] with $\vee$.
[2] This condition was not considered in [9].

2.3 Modal Deductive Databases and Data Complexity

An MDatalog/eMDatalog program clause is either a rule or a fact. A *rule* is a program clause containing $\leftarrow$, while a *fact* does not contain $\leftarrow$. Note that an MDatalog fact is a ground formula of the form $\Box^h A$, where $h \geq 0$ and A is a formula of the form $\Box E$, $\Diamond E$, or E with E being a classical atom. Observe also that an eMDatalog fact is a positive ground formula not containing $\vee$.

A predicate p is defined by an MDatalog/eMDatalog program clause φ if p appears in φ in the left hand side of all the occurrences of $\leftarrow$.

A modal deductive database can be specified by an MDatalog/eMDatalog program. It can be divided into two parts: an *extensional part* and an *intensional part*. The extensional part consists of facts defining so called *extensional predicates*. The intensional part consists of rules and facts defining so called *intensional predicates*, which are not extensional predicates.

When measuring the "data complexity" of a query language for deductive databases, the query to the program specifying the database is grouped with the intensional part of the database and treated as a fixed *query*, while the extensional part of the database is treated as input. Suppose that we are given an MDatalog (resp. eMDatalog) program P representing a modal deductive database and a query $\exists\overline{x}\,\varphi(\overline{x})$. Denote the extensional part of the database by D and the intensional part by P'. We call the pair $(P', \exists\overline{x}\,\varphi(\overline{x}))$ an *MDatalog* (resp. *eMDatalog*) *query* and D an *extensional MDatalog* (resp. *eMDatalog*) *database instance* (*edb instance* for short).

We say that the *data complexity* of MDatalog (resp. eMDatalog) in a modal logic L is in a complexity class $\mathcal{C}$ if the recognition problem of checking whether $P' \cup D \models_L \varphi(\overline{c})$ with D taken as input is in the complexity class $\mathcal{C}$ for every MDatalog (resp. eMDatalog) query $(P', \exists\overline{x}\,\varphi(\overline{x}))$ and every tuple $\overline{c}$ of constant symbols.

We say that the data complexity of MDatalog (resp. eMDatalog) is complete in a complexity class $\mathcal{C}$, or $\mathcal{C}$*-complete*, if it is in $\mathcal{C}$ and there *exist* an MDatalog (resp. eMDatalog) query $(P', \exists\overline{x}\,\varphi(\overline{x}))$ and a tuple $\overline{c}$ of constant symbols such that the problem of checking whether $P' \cup D \models_L \varphi(\overline{c})$ with D taken as input is $\mathcal{C}$-complete.

It is not clear whether the data complexity of eMDatalog is the same as the data complexity of MDatalog in every modal logic. The problem is that when transforming an eMDatalog program to an MDatalog program, we introduce new rules, which causes that the resulting "query" depends on the input. For this reason, we will study the data complexity of both MDatalog and eMDatalog.

3 Simple Cases

In this section, we show that the data complexity of MDatalog and eMDatalog in $K5$, $KD5$, $K45$, $KD45$, $KB5$, and $S5$ is PTIME-complete, and in $K4$, $KD4$, and $S4$ is in PSPACE.

Let $(P, \exists\overline{x}\,\varphi(\overline{x}))$ be a fixed MDatalog/eMDatalog query, $\overline{c}$ be a fixed tuple of constant symbols for substituting $\overline{x}$, and D be an input *edb* instance with size n.

Let P' be the set of all ground instances of program clauses of P using the constant symbols occurring in P, $\overline{c}$, and D. It is easily seen that the size of P' is bounded by a polynomial of n. Observe that $P \cup D \models_L \varphi(\overline{c})$ iff $P' \cup D \models_L \varphi(\overline{c})$ iff $P' \cup D \cup \{\neg\varphi(\overline{c})\}$ is L-unsatisfiable. The latter set can be treated as a set of propositional formulas. It consists of so called *Horn formulas.* By the results of [3,1], checking whether $P' \cup D \cup \{\neg\varphi(\overline{c})\}$ is L-unsatisfiable is decidable in PTIME (w.r.t. n) for $L \in \{K5, KD5, K45, KD45, KB5, S5\}$.[3] By Ladner [7], for $L \in \{K4, KD4, S4\}$, checking whether $P' \cup D \cup \{\neg\varphi(\overline{c})\}$ is L-unsatisfiable is decidable in PSPACE (the Horn property is not important here). We arrive at:

Theorem 1. *The data complexity of MDatalog and eMDatalog in K5, KD5, K45, KD45, KB5, and S5 is PTIME-complete.*

The lower bound follows from that the data complexity of Datalog is complete in PTIME (see, e.g., [6]).

Lemma 1. *The data complexity of MDatalog and eMDatalog in K4, KD4, and S4 is in PSPACE.*

4 The Data Complexity of MDatalog in K, $K4$, $KD4$, $S4$

In this section, we show that the data complexity of MDatalog and eMDatalog in $K4$, $KD4$, and $S4$ is PSPACE-complete, and in K is coNP-complete.

Lemma 2. *Every finite set X of propositional modal Horn clauses can be transformed in PTIME to a set Y of propositional modal Horn clauses such that:*

- *Y contains at most one negative clause, which is of the form $\leftarrow p$;*
- *each non-negative clause of Y contains no more than 3 literals;*
- *the modal depth of Y is the same as the modal depth of X;*
- *Y is L-satisfiable iff X is L-satisfiable, for any normal modal logic L.*

Proof. Let p be a fresh primitive proposition. Replace each negative clause $\varphi_i = \Box^s(\leftarrow B_1, \ldots, B_k)$ of X by the following clauses:

$$\Box^s(p_{i,s} \leftarrow B_1, \ldots, B_k), \quad \Box^{s-1}(p_{i,s-1} \leftarrow \Diamond p_{i,s}), \quad \ldots, \quad p \leftarrow \Diamond p_{i,1}$$

where $p_{i,s}, \ldots, p_{i,1}$ are fresh primitive propositions. (If $s = 0$ then φ_i is replaced by $p \leftarrow B_1, \ldots, B_k$.) Then add to the obtained set the clause $\leftarrow p$. Denote the resulting set by X'.

Next, replace each non-negative clause $\varphi_i = \Box^s(A \leftarrow B_1, \ldots, B_k)$ of X', where $k > 2$, by the following clauses:

$$\begin{array}{l} \Box^s(p_{i,2} \leftarrow B_1, B_2) \\ \Box^s(p_{i,3} \leftarrow p_{i,2}, B_3) \\ \ldots \\ \Box^s(p_{i,k} \leftarrow p_{i,k-1}, B_k) \\ \Box^s(A \leftarrow p_{i,k}) \end{array}$$

[3] The definitions of Horn clauses in [3,1] are more restrictive than our definition of Horn formulas [8]. However, every set of Horn formulas can be transformed in polynomial time to a set of Horn clauses that preserves satisfiability.

where $p_{i,2}, \ldots, p_{i,k}$ are fresh primitive propositions. Let Y be the resulting set. It is easy to verify that Y satisfies the assertions of the lemma.

Corollary 1. *Let X be a set of propositional modal Horn clauses such that: the modal depth of X is not greater than 2, X contains at most one negative clause, which is of the form $\leftarrow p$, and each non-negative clause of X contains no more than 3 literals. Then the problem of checking whether X is L-satisfiable is NP-complete for $L = K$ and PSPACE-complete for $L \in \{K4, KD4, S4\}$.*

Proof. This corollary immediately follows from Lemma 2 and the results of [11,1] that the satisfiability problem of sets of propositional modal Horn clauses with modal depth bounded by $k \geq 2$ in K is NP-complete, in $K4$, $KD4$, and $S4$ is PSPACE-complete.

Lemma 3. *Let X be as in Corollary 1 with $\psi = \leftarrow q$ being the only negative clause. Then there exist an MDatalog query (P, φ), where P does not depend on X and φ is a positive ground formula depending only on ψ, and an* edb *instance D with size in polynomial order in the size of X such that X is L-satisfiable iff $P \cup D \not\models_L \varphi$, where L is any normal modal logic.*

Proof. Each non-negative clause of X is of the form $\Box^s(A \leftarrow B_1, \ldots, B_k)$, where $0 \leq s \leq 2$, $0 \leq k \leq 2$, and $A, B_1, \ldots, B_k$ are of the form p, $\Box p$, or $\Diamond p$. Let K be the number of such possible forms (i.e. $K = 3 \times 3 \times (1 + 3 + 3 \times 3)$).

Suppose that primitive propositions occurring in X are numbered from 1 to n and denoted by $p_1, \ldots, p_n$. We represent each non-negative clause ξ of X by a first-order program clause representing the form of ξ plus a ground first-order atom indicating ξ. For simplicity, we illustrate this using a clause $\xi = \Box\Box(p_i \leftarrow \Box p_j, \Diamond p_k)$. Let $1 \leq t \leq K$ be the number indicating the form of this clause. The clause ξ is then represented by $\Box\Box(p(x) \leftarrow \Box p(y), \Diamond p(z), clause(t, x, y, z))$ and $\Box\Box clause(t, i, j, k)$. Note that the program clause depends only on the form of ξ, while the atom indicates ξ. In this way, the set of non-negative clauses of X is represented by an MDatalog program P and a set D of ground first-order atoms. Furthermore, we can assume that P does not depend on X, as we can include in P an appropriate program clause for *every* $1 \leq t \leq K$.

If $\psi = \leftarrow p_i$ then let $\varphi = p(i)$. It is clear that X is L-satisfiable iff $P \cup D \cup \{\neg\varphi\}$ is L-satisfiable, and iff $P \cup D \not\models_L \varphi$, where L is any normal modal logic.

Theorem 2. *The data complexity of MDatalog and eMDatalog in K4, KD4, and S4 is PSPACE-complete.*

Proof. By Lemma 1, the data complexity of MDatalog and eMDatalog in $K4$, $KD4$, and $S4$ is in PSPACE. Let X be a set of propositional modal Horn clauses as in Lemma 3. By Corollary 1, the problem of checking L-satisfiability of X for $L \in \{K4, KD4, S4\}$ is PSPACE-complete. By Lemma 3, this problem is reducible in polynomial time to a problem of answering a fixed MDatalog query (P, φ) w.r.t. an *edb* instance D depending on X (here, without loss of generality, assume that φ is fixed). Hence, the data complexity of MDatalog and eMDatalog in $K4$, $KD4$, and $S4$ is complete in PSPACE.

Theorem 3. *The data complexity of MDatalog and eMDatalog in K is coNP-complete.*

Proof. We first show that the data complexity of eMDatalog in K is in coNP. Let (P, φ) be a fixed eMDatalog query, where φ is a positive ground formula, and D be an eMDatalog *edb* instance. To answer the query w.r.t. the input D in the logic K is to check whether $P \cup D \models_K \varphi$, or equivalently, whether $P \cup D \cup \{\neg\varphi\}$ is K-unsatisfiable.

Let n be the size of D and c be the modal depth of $P \cup \{\neg\varphi\}$. Let P' be the set of all ground instances of clauses of P using the constant symbols occurring in P, φ, D. The size of P' is bounded by a polynomial of n. Observe that $P \cup D \cup \{\neg\varphi\}$ is K-satisfiable iff $P' \cup D \cup \{\neg\varphi\}$ is K-satisfiable.

Consider the process of constructing a K-model graph for $P' \cup D \cup \{\neg\varphi\}$. At the beginning, the model graph contains only the actual world τ with $P' \cup D \cup \{\neg\varphi\}$ in the negative normal form as the content. Then for every world w and every formula ψ from the content of w, we realize ψ at w as follows. If $\psi = \zeta_1 \vee \ldots \vee \zeta_k$ then nondeterministically choose some ζ_i and add ζ_i to the content of w. If $\psi = \Diamond\zeta$ then create a new empty world u, connect w to u via the accessibility relation, and add ζ to the content of u. If $\psi = \Box\zeta$ then add ζ to the content of every world accessible from w. If at the end we obtain a model graph which is saturated (in the sense that, for every w, every formula ψ in the content of w has been realized at w) and is consistent (in the sense that no world contains both p and $\neg p$ for some primitive proposition p), then $P' \cup D \cup \{\neg\varphi\}$ is K-satisfiable. Observe that if a world w in the constructed model graph is inconsistent (i.e. w contains both p and $\neg p$ for some p), then the path from τ to w via the accessibility relation is not longer than c, since D contains only positive formulas. This means that when checking consistency of the constructed model graph, we need to pay attention only to worlds not far away from τ than c edges. The total size of that fragment of the constructed model graph is bounded by a polynomial of n. Hence, checking whether $P' \cup D \cup \{\neg\varphi\}$ is K-satisfiable can be nondeterministically done in polynomial time. Consequently, the problem of answering the query (P, φ) w.r.t. the input D in the logic K is in the coNP class.

Analogously as in the proof of Theorem 2, using Corollary 1 and Lemma 3 we conclude that the data complexity of MDatalog and eMDatalog in K is complete in coNP.

5 The Data Complexity of eMDatalog in *KD*, *T*, *KB*, *KDB*, and *B*

In this section, we show that the data complexity of MDatalog and eMDatalog in *KD*, *T*, *KB*, *KDB*, and *B* is complete in PTIME. For this aim, we first present an algorithm of constructing a "least" *L*-model for a given eMDatalog program, where $L \in \{KD, T, KDB, B\}$.

We say that a propositional Kripke model M is less than or equal to M' if for every positive propositional formula φ, if $M \models \varphi$ then $M' \models \varphi$. M is called a *least L-model* of an eMDatalog program P if M is an L-model of P and M is less than or equal to every L-model of P. Observe that if M is a least L-model of P and φ is a positive propositional formula then $P \models_L \varphi$ iff $M \models_L \varphi$.

In the algorithm given below, as a data structure we have a model graph $M = \langle W, \tau, R, H \rangle$ and a binary relation R' being the skeleton of R. We sometimes refer to M as the propositional model $\langle W, \tau, R, h \rangle$, with $h(x) = \{E \mid E$ is a ground classical atom belonging to $H(x)\}$. We write $M, u \models \varphi$ to denote that φ is true at u in the *model* M.

We will use a procedure $CreateEmptyTail_L(x_0)$ defined as: Add an infinite chain of new empty worlds $x_1, x_2, \ldots$ to W, set $R' = R' \cup \{(x_i, x_{i+1}) \mid i \geq 0\}$, and set R to the least extension of R' that satisfies all L-frame restrictions. (Note that the chain can be coded as a finite chain, which will be dynamically expanded when necessary).

Algorithm 1
Input: A ground eMDatalog program P in $L \in \{KD, T, KDB, B\}$
treated as a set of propositional formulas.
Output: A least L-model $M = \langle W, \tau, R, h \rangle$ of P.

```
1. Set $W = \{\tau\}$, $H(\tau) = P$, $R' = \emptyset$.
   $CreateEmptyTail_L(\tau)$.
2. For every $u \in W$ and for every $\varphi \in H(u)$:
   (a) If $\varphi = (\psi \leftarrow \zeta)$ and $M, u \models \zeta$ then set $H(u) = H(u) \cup \{\psi\}$.
   (b) If $\varphi = \psi \wedge \zeta$ then set $H(u) = H(u) \cup \{\psi, \zeta\}$.
   (c) If $\varphi = \Box\psi$ then for every $v \in W$ s.t. $R(u, v)$ set $H(v) = H(v) \cup \{\psi\}$.
   (d) If $\varphi = \Diamond\psi$ and $\neg(\exists x\, R'(u, x) \wedge \psi \in H(x))$ then:
           Let $u_\psi$ be a new world.
           Set $W = W \cup \{u_\psi\}$, $H(u_\psi) = \{\psi\}$, $R' = R' \cup \{(u, u_\psi)\}$.
           $CreateEmptyTail_L(u_\psi)$.
3. While some change occurred, repeat step 2.
```

This algorithm is very similar to Algorithm 5.1 in [8], which constructs a least L-model for a given positive propositional modal logic program. The following lemma can be proved as done for Algorithm 5.1 in [8].

Lemma 4. *The above algorithm always terminates and the constructed model M is a least L-model of P.*

Let $M = \langle W, \tau, R, h \rangle$ be a Kripke model. Define the distance from τ to a world $w \in W$ via R to be the length of a shortest path from τ to w via R (undefined if there does not exist such a path). For $k \geq 0$, we define $M_{|k}$ to be the model obtained from M by restricting it to the worlds with the distance from τ via R not greater then k. The following lemma can be proved easily.

Lemma 5. *Let $M = \langle W, \tau, R, h \rangle$ be a Kripke model and φ a formula with modal depth not greater than k. Then $M \models \varphi$ iff $M_{|k} \models \varphi$.*

If we are given an eMDatalog program P representing a modal deductive database and a positive ground formula φ to check whether $P \models_L \varphi$, then instead of constructing a least L-model M of P to check whether $M \models \varphi$ we can construct only $M_{|k}$ and check whether $M_{|k} \models \varphi$ where k is a number not less than the modal depth of φ. In the following we show that such a restricted model $M_{|k}$ can be constructed without having the whole model M.

Let φ be a positive ground formula and ψ be a subformula of φ with a fixed position. We define the *modal context of ψ in φ* to be the sequence of modal operators occurring in φ such that ψ is under the scope of them. For example, the modal context of $\Box(r \wedge s)$ in $\Diamond(\Box p \wedge \Diamond(\Diamond q \wedge \Box(r \wedge s)))$ is $\Diamond\Diamond$, which consists of the first and the second $\Diamond$. We call a sequence of modal operators a *modality* and denote the empty modality by ε.

For $L \in \{T, KDB, B\}$, let the *set of rules for shrinking modalities in L* consist of: $\Box \to \varepsilon$ if $L \in \{T, B\}$, $\Box\Box \to \varepsilon$ and $\Diamond\Box \to \varepsilon$ if $L \in \{KDB, B\}$. Note that the first rule corresponds to axiom T, while the two latter rules correspond to axiom B. A modality $\triangle$ is *reducible to ε* in $L \in \{T, KDB, B\}$ if ε is derivable from $\triangle$ using the rules for shrinking modalities in L.

Lemma 6. *Consider Algorithm 1 and a moment when a formula ψ is added to $H(x)$. Suppose that the distance from τ to x via R' is greater than the modal depth of every rule of P. Then:*

- *there exists $\Box\psi \in H(y)$ s.t. $R'(y, x)$ holds; or*
- *there exists $\Diamond\psi \in H(y)$ s.t. $x = y_\psi$ (i.e. x is created from y using $\Diamond\psi$); or*
- *before adding ψ to $H(x)$ there exists already $\psi' \in H(x)$ such that ψ is a subformula of ψ' under a modal context which is reducible to ε in L.*

Proof. We prove this lemma by induction on the number of steps needed to add ψ to $H(x)$. The only non-trivial case is when ψ is added to $H(x)$ at Step 2c with $x = v$, $L \in \{KDB, B\}$ and $R'(v, u)$ holds. Consider that case. Applying the induction hypothesis for the moment when $\varphi = \Box\psi$ is added to $H(u)$, we obtain that $\Diamond\varphi \in H(v)$ or $\Box\varphi \in H(v)$ or before adding φ to $H(u)$ there exists already $\varphi' \in H(u)$ such that φ is a subformula of φ' under a modal context which is reducible to ε in L. By repeatedly applying the induction hypothesis for the moment when φ' is added to $H(u)$, we derive that there exists a formula $\xi \in H(u)$ such that φ is a subformula of ξ under a modal context which is reducible to ε and ξ was added to $H(u)$ because $\Diamond\xi \in H(v)$ or $\Box\xi \in H(v)$. It is easily seen that the inductive assertion immediately follows.

Lemma 7. *Consider Algorithm 1. Suppose that $\varphi \in H(u)$ and the distance from τ to u via R' is greater than the modal depth of every rule of P. Then every subformula ψ of φ under a modal context which is reducible to ε will be added to $H(u)$.*

Proof. By induction on the structure of φ.

We now present a modified version of Algorithm 1. The modifications are highlighted by another font.

Algorithm 2

```
Input: A ground eMDatalog program $P$ in $L \in \{KD, T, KDB, B\}$
       treated as a set of propositional formulas.
       A number $k$ greater than the modal depth of every rule of $P$.
Output: A model $M = \langle W, \tau, R, h \rangle$.

1. Set $W = \{\tau\}$, $H(\tau) = P$, $R' = \emptyset$.
   $CreateEmptyTail_L(\tau)$.
2. For every $u \in W$ with $H(u)$ not empty and for every $\varphi \in H(u)$:
   (a) If $\varphi = (\psi \leftarrow \zeta)$ and $M, u \models \zeta$ then set $H(u) = H(u) \cup \{\psi\}$.
   (b) If $\varphi = \psi \wedge \zeta$ then set $H(u) = H(u) \cup \{\psi, \zeta\}$.
   (c) If $\varphi = \Box\psi$ then for every $v \in W$ s.t. $R(u, v)$ set $H(v) = H(v) \cup \{\psi\}$.
   (d) If $\varphi = \Diamond\psi$ and $\neg(\exists x\ R'(u, x) \wedge \psi \in H(x))$ and the path from $\tau$ to $u$ via
       $R'$ is shorter than $k$ then:
           Let $v$ be a new world.
           Set $W = W \cup \{v\}$, $H(v) = \{\psi\}$, $R' = R' \cup \{(u, v)\}$.
           $CreateEmptyTail_L(v)$.
   (e) If $L \in \{T, KDB, B\}$ and the path from $\tau$ to $u$ via $R'$ has length $k$ then, for
       every subformula $\psi$ of $\varphi$ under a modal context which is reducible to $\varepsilon$ in
       $L$, set $H(u) = H(u) \cup \{\psi\}$.
3. While some change occurred, repeat step 2.
```

Lemma 8. *Let P be a ground eMDatalog program s.t. the modal depths of the rules of P are not greater than k. Let M and M' be respectively the outputs of Algorithm 1 and Algorithm 2 for P in $L \in \{KD, T, KDB, B\}$. Then $M_{|k}$ is isomorphic with M'.*

This lemma immediately follows from Lemmas 6 and 7.

Theorem 4. *The data complexity of MDatalog and eMDatalog in $L \in \{KD, T, KDB, B\}$ is PTIME-complete.*

Proof. Since the data complexity of Datalog is complete in PTIME (see, e.g., [6]), it suffices to show that the data complexity of eMDatalog in L is in PTIME.

Let (P', φ) be a fixed eMDatalog query, where φ is a positive ground formula, and D be an eMDatalog *edb* instance. To answer the query w.r.t. the input D in L is to check whether $P' \cup D \models_L \varphi$. Let P'' be the set of all ground instances of clauses of P' using the constant symbols occurring in P', φ, D. Observe that $P' \cup D \models_L \varphi$ iff $P'' \cup D \models_L \varphi$.

Let $P = P'' \cup D$ and let k be the maximum of the modal depths of P' and φ plus 1. Note that k is fixed. Let M be the result of the execution of Algorithm 2 for P and k. By Lemmas 4, 5, and 8, we have that $P'' \cup D \models_L \varphi$ iff $M \models \varphi$.

Let n be the size of D. As the query (P', φ) is fixed, the size of P'' is bounded by a polynomial of n. The size of M and the number of steps needed to construct M are bounded by a polynomial in the size of P''. Checking whether $M \models \varphi$ can be done in polynomial time in the sizes of M and φ. Totally, checking whether $P' \cup D \models_L \varphi$ can be done in polynomial time in the size of the input D, and hence the data complexity of eMDatalog in L is in PTIME.

The remaining logic KB is an *almost serial* (see [9]) modal logic corresponding to B. Using the techniques of [8,9], we can derive the following result.

Corollary 2. *The data complexity of MDatalog and eMDatalog in* KB *is PTIME-complete.*

6 Conclusions

We have proved that, for L being any one of the 15 basic monomodal logics, the data complexity of MDatalog and eMDatalog in L is the same as the complexity of the unsatisfiability problem in L of the propositional modal Horn fragment with modal depth bounded by $k \geq 2$. This result is a bit surprising, as MDatalog and eMDatalog are fragments of first-order modal logics and can contain data with unbounded modal depth. The eMDatalog language is an interesting fragment of modal logic that has low complexity in KD, T, KB, KDB, and B. On the other hand, the PSPACE-complete data complexity of MDatalog in $K4$, $KD4$, and $S4$ shows that these latter modal logics (and their multimodal extensions) are hard in the view of deductive databases (unless PSPACE = PTIME).

Acknowledgements. I would like to thank the reviewers for useful comments.

References

1. C.C. Chen and I.P. Lin. The computational complexity of the satisfiability of modal Horn clauses for modal propositional logics. *Theor. Comp. Sci.*, 129:95–121, 1994.
2. M.J. Cresswell and G.E. Hughes. *A New Introduction to Modal Logic*. Routledge, 1996.
3. L. Fariñas del Cerro and M. Penttonen. A note on the complexity of the satisfiability of modal Horn clauses. *Logic Programming*, 4:1–10, 1987.
4. M. Fitting and R.L. Mendelsohn. *First-Order Modal Logic*. Springer, 1998.
5. G. Gottlob, E. Grädel, and H. Veith. Linear time datalog and branching time logic. In *Logic-Based Artif. Int.*, pages 443–467. Kluwer Academic Publishers, 2000.
6. Ch. Koch and S. Scherzinger. Lecture notes on database theory. `http://www-db.cs.unisb.de/teaching/dbth0506/slides/dbthdatalog2.pdf`.
7. R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6:467–480, 1977.
8. L.A. Nguyen. Constructing the least models for positive modal logic programs. *Fundamenta Informaticae*, 42(1):29–60, 2000.
9. L.A. Nguyen. The modal query language MDatalog. *Fundamenta Informaticae*, 46(4):315–342, 2001.
10. L.A. Nguyen. On modal deductive databases. In J. Eder, H.-M. Haav, A. Kalja, and J. Penjam, editors, *Proc. of ADBIS 2005, LNCS 3631*, pages 43–57. Springer, 2005.
11. L.A. Nguyen. On the complexity of fragments of modal logics. In R. Schmidt, I. Pratt-Hartmann, M. Reynolds, and H. Wansing, editors, *Advances in Modal Logic - Volume 5*, pages 249–268. King's College Publications, 2005.

The Complexity of Counting Functions with Easy Decision Version[⋆]

Aris Pagourtzis[1] and Stathis Zachos[1,2]

[1] Department of Computer Science, School of ECE,
National Technical University of Athens, Greece
{pagour, zachos}@cs.ntua.gr
[2] CIS Department, Brooklyn College, CUNY

Abstract. We investigate the complexity of counting problems that belong to the complexity class #P and have an easy decision version. These problems constitute the class #PE which has some well-known representatives such as #Perfect Matchings, #DNF-Sat, and NonNegative Permanent. An important property of these problems is that they are all #P-complete, in the Cook sense, while they cannot be #P-complete in the Karp sense unless P = NP.

We study these problems in respect to the complexity class TotP, which contains functions that count the number of *all* paths of a PNTM. We first compare TotP to #P and #PE and show that $\mathsf{FP} \subseteq \mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{P}$ and that the inclusions are proper unless P = NP.

We then show that several natural #PE problems — including the ones mentioned above — belong to TotP. Moreover, we prove that TotP is exactly the Karp closure of self-reducible functions of #PE. Therefore, all these problems share a remarkable structural property: for each of them there exists a polynomial-time nondeterministic Turing machine which has as many computation paths as the output value.

1 Introduction

The complexity class #P, introduced by Valiant [Val79a], is the class of functions that count the number of accepting paths of a polynomial-time nondeterministic Turing machine (PNTM). This class contains several interesting problems, in particular counting variants of classical NP-search problems. Some complete problems for this class, and thus hard to compute, are #Sat, #Cliques, etc. A noteworthy fact is that, while for these problems the decision (existence) version is NP-complete, there are other #P-complete problems which have easy (polynomial-time decidable) existence versions: #Perfect Matchings, #DNF-Sat, and many more.

In this paper we investigate the complexity of such "hard-to-count-easy-to-decide" problems (a term borrowed from [DHK00]). A key observation is that these latter problems are #P-complete under Cook reductions — but cannot

[⋆] Research supported in part by grant EΔ285, under the ΠENEΔ 2003 programme of the General Secretariat of Research and Technology of Greece.

be #P-complete under Karp reductions. It turns out that Cook reductions blur structural differences between counting problems and complexity classes. The reason is that #P as well as several other classes is not closed under Cook reductions (under reasonable assumptions). Therefore, in an effort to characterize these problems, we look for subclasses of #P— closed under Karp reduction — for which there is a possibility that (some of) these problems are Karp-complete.

The complexity class TotP was introduced in [KPSZ98] as the class of functions that count the total number of computational paths of a PNTM; it was proved that this new class is equivalent to #P under Cook reductions. On the other hand, it was shown in [KPZ99] that #P does not reduce to TotP under Karp reductions, that is, TotP is a proper subset of #P unless $\mathsf{P} = \mathsf{NP}$.

A second class of functions with similar properties was introduced in [Pag01]. This class was called #PE (for #P "Easy") since it contains all functions f of #P such that for any input x the question '$f(x) > 0$?' is polynomial-time decidable. By definition, #PE contains #PERFECT MATCHINGS and #DNF-SAT but not #SAT unless P=NP.

Here we explore the relations among #P, #PE, and TotP, and we clarify which #PE problems are contained in TotP. Our most important results can be summarized as follows.

– $\mathsf{FP} \subseteq \mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{P}$. Inclusions are proper unless $\mathsf{P} = \mathsf{NP}$; that is, TotP, #PE, and #P are not Karp-equivalent unless $\mathsf{P} = \mathsf{NP}$. On the other hand, $\mathsf{FP}^{\mathsf{TotP}[1]} = \mathsf{FP}^{\#\mathsf{PE}[1]} = \mathsf{FP}^{\#\mathsf{P}[1]}$; that is, TotP, #PE, and #P are Cook[1]-equivalent.
– #PERFECT MATCHINGS, #DNF-SAT, #NONCLIQUES, and NONNEGATIVE PERMANENT, as well as some related problems, are in TotP, in fact TotP-complete under Cook[1] reduction.
– TotP is exactly the Karp closure of self-reducible functions of #PE. This implies that all these problems possess a remarkable structural property: for each of them there exists a PNTM with as many computation paths as the output value.

2 Preliminaries and Definitions

Our model of computation is the polynomial-time bounded nondeterministic Turing machine (PNTM), i.e., there is some polynomial p so that for any input x, all computation paths have length at most $p(|x|)$, where $|x|$ is the length of the input. A counting function $\Sigma^* \to \mathbb{N}$ is associated with a PNTM M:

$$\mathrm{acc}_M(x) = \#\text{accepting paths of } M \text{ on input } x$$

In his seminal paper [Val79a] Valiant introduced the class of counting functions #P:

$$\#\mathsf{P} = \{\mathrm{acc}_M \mid M \text{ is a PNTM}\}$$

Equivalently, $\#\mathsf{P}$ can be seen as the class of functions that count the number of solutions of NP-search problems.

$\#\mathsf{P}$ is a class of very high complexity: Toda showed [Tod91] that $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}[1]}$, where PH stands for the Polynomial-Time Hierarchy.

We consider functional oracles and use P^g to denote the class of languages that can be decided by a polynomial-time deterministic Turing machine (PDTM) with access to an oracle function g. For a function class $\mathcal{G}$, $\mathsf{P}^{\mathcal{G}} = \bigcup_{g\in\mathcal{G}} \mathsf{P}^g$. The function classes FP^g and $\mathsf{FP}^{\mathcal{G}}$ are defined analogously, considering PDTMs with output equipped with a function oracle.

For complexity classes of functions $\mathcal{F}$ and $\mathcal{G}$, $\mathcal{F} - \mathcal{G}$ denotes the class of functions that can be expressed as the difference of one function in $\mathcal{F}$ and one function in $\mathcal{G}$.

Reductions between functions can be defined in a similar manner to the Cook/Turing and Karp/many-one reductions between languages. Valiant [Val79a] presented a Cook reduction with one oracle call (that is, a Cook[1] reduction) from any problem in $\#\mathsf{P}$ to #Perfect Matchings. Since then, Cook[1] reductions between functions have been widely used (e.g., in [Kre88, KST89, TW92]); Krentel in [Kre88] calls such reductions *metric*. The notion of Karp/many-one reducibility between functions has also been often used in the literature [Wag86, Tor88, Vol94]. When referring to counting functions corresponding to NP-search problems, this kind of reduction is often called *parsimonious*, a name credited to Simon [Sim75]. We shall use the terms "Cook", "Cook[1]", and "Karp", as shortcuts for "poly-time Turing", "poly-time 1-Turing",and "poly-time many-one", respectively.

Definition 1. *Polynomial-time reductions between functions:*

- *Cook (poly-time Turing)* $f \leq^p_{\mathrm{T}} g : f \in \mathsf{FP}^g$.
- *Cook[1] (poly-time 1-Turing)* $f \leq^p_{1-\mathrm{T}} g : f \in \mathsf{FP}^{g[1]}$.
- *Karp (poly-time many-one)* $f \leq^p_{\mathrm{m}} g : \exists\, h \in \mathsf{FP}, \forall x\ f(x) = g(h(x))$.

All three reductions are reflexive and transitive, and the following holds for all functions f, g:

$$f \leq^p_{\mathrm{m}} g \Rightarrow f \leq^p_{1-\mathrm{T}} g \Rightarrow f \leq^p_{\mathrm{T}} g$$

We say that $\mathcal{G}$ is *closed under* $\leq_r$ iff $f \leq_r g \wedge g \in \mathcal{G} \Rightarrow f \in \mathcal{G}$.

For a function class $\mathcal{F}$, a function g is called $\mathcal{F}$-hard under a reduction $\leq_r$ if for all $f \in \mathcal{F}$, $f \leq_r g$. A function g is called $\mathcal{F}$-complete under $\leq_r$ if it is $\mathcal{F}$-hard under $\leq_r$ and $g \in \mathcal{F}$.

There is an important difference between Karp and Cook reductions. Namely, most of the known function classes that can be defined via PNTMs (including $\#\mathsf{P}$) are closed under $\leq^p_{\mathrm{m}}$. On the other hand, the class $\mathsf{FP}^{\#\mathsf{P}[1]}$ by definition Cook[1]-reduces to $\#\mathsf{P}$, while it is not contained in $\#\mathsf{P}$ (unless PH collapses—see [TW92]); this means that $\#\mathsf{P}$ is not closed under $\leq^p_{1-\mathrm{T}}$ unless PH collapses.

Therefore, we can say that Cook reductions blur structural differences between complexity classes of functions [KPZ99].[1]

3 Appropriate Classes for "Hard-to-Count-Easy-to-Decide" Problems

The difference between Karp and Cook reductions mentioned in the previous section is well reflected in the behavior of #P-complete problems. While there exist problems which are #P-complete under both reductions (namely, the counting versions of most NP-complete problems), there also exist problems which are #P-complete only under the Cook reduction (unless P=NP). These latter problems usually have a decision version in P and are sometimes called "hard-to-count-easy-to-decide" problems. Some examples: #Perfect Matchings, #DNF-Sat, #NonCliques.

We would like to determine the exact complexity of such problems. We are therefore looking for reasonably defined subclasses of #P that contain them; ideally we would like to show that (some of) these problems are complete for some subclass $\mathcal{F}$ of #P under Karp reduction (or under a reduction for which $\mathcal{F}$ is closed). Two recently introduced classes are good candidates: #PE [Pag01] and TotP [KPSZ98].

3.1 The Classes #PE and TotP

For each function $f \in \Sigma^* \rightarrow \mathbb{N}$ we define a related language:

$$L_f = \{x \mid f(x) > 0\}$$

Note that for function problems this language represents a natural decision version of the problem. In particular, if a function f corresponds to the counting version of a search problem (i.e., f counts how many solutions are there for a given instance) then L_f corresponds to the existence version.

The class #PE contains functions of #P whose related language is in P (i.e., the question "$f(x) > 0$?" is polynomial-time decidable). In other words, #PE by definition contains all *hard-to-count-easy-to-decide* problems.

The definition of TotP involves a function associated with every PNTM M:

$$\mathrm{tot}_M(x) = (\#\text{paths of } M \text{ on input } x) - 1$$

Remark 1. The 'minus one' in the definition of tot_M above was introduced so that the function can have a zero value.

TotP is defined as the class of all tot_M functions:

$$\mathsf{TotP} = \{\mathrm{tot}_M \mid M \text{ is a PNTM}\}$$

TotP and #PE have several similarities — and some differences. We next investigate the relation between them, as well as their relation to #P.

[1] This is true for language classes as well. For example, the CNF-Sat problem is complete under Cook reduction both for NP and P^{NP}.

3.2 Inclusions Among TotP, #PE, and #P

Proposition 1. *1.* $\#\mathsf{P} \subseteq \mathsf{TotP} - \mathsf{FP}$ *[KPSZ98].*
2. $\mathsf{FP} \subseteq \mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{P}$*. Inclusions are proper, unless* $\mathsf{P} = \mathsf{NP}$*.*

Proofs are omitted due to lack of space; they will be included in the full version of the paper.

The fact that inclusions in Proposition 1.2 are proper unless $\mathsf{P} = \mathsf{NP}$, together with the fact that TotP, #PE, and #P, are closed under Karp reduction, implies the following.

Corollary 1. TotP*,* #PE*, and* #P *are not Karp-equivalent unless* $\mathsf{P} = \mathsf{NP}$*.*

It is easy to see that for any polynomially length-bounded[2] function class $\mathcal{F}$ it holds $\mathcal{F} - \mathsf{FP} \subseteq \mathsf{FP}^{\mathcal{F}}$. Hence, Proposition 1 can be used to show that:

Corollary 2. $\mathsf{FP}^{\mathsf{TotP}[1]} = \mathsf{FP}^{\#\mathsf{PE}[1]} = \mathsf{FP}^{\#\mathsf{P}[1]}$*. That is,* TotP*,* #PE*, and* #P *are Cook[1]-equivalent.*

A further implication of the above, combining with the result of Toda [Tod91], is that $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{TotP}[1]} = \mathsf{P}^{\#\mathsf{PE}[1]}$.

The above corollaries imply that under Karp reduction #P-complete, #PE-complete and TotP-complete problems constitute disjoint classes, unless $\mathsf{P} = \mathsf{NP}$, while under Cook[1] reduction TotP-complete problems are contained in #PE-complete problems which are contained in #P-complete problems.

Hence, the information that a problem is #P-complete under Cook[1] reduction does not suffice in order to classify the problem well. For example, #Perfect Matchings is a known #P-complete problem under Cook[1] reduction. It is not hard to observe that it is also #PE-complete. Could it even be TotP-complete? In the next section we answer this question in the affirmative.

4 Some #P-Complete Problems Are Also TotP-Complete

Theorem 1. #Perfect Matchings *is* TotP*-complete under Cook[1] reduction.*

Proof. #Perfect Matchings is certainly TotP-hard under Cook[1]-reduction, since it is #P-complete under this reduction [Val79b] and $\mathsf{TotP} \subseteq \#\mathsf{P}$. It remains to prove that #Perfect Matchings belongs to TotP.

We construct a PNTM M which, for any input graph G, makes nondeterministic choices so that the number of computation paths is equal to the number of perfect matchings of G plus one — hence $\mathrm{tot}_M(G) = \#\mathrm{PM}(G)$. We present the description of M in the form of a nondeterministic algorithm.

[2] A function is polynomially length-bounded if $\exists$ polynomial p s.t. for all x, $|f(x)| \leq p(|x|)$.

```
Nondeterministic algorithm for #PerfectMatchings (#PM)
(* For an input graph $G = (V, E)$, the computation tree has $\#\mathrm{PM}(G) + 1$ leaves *)
begin
   if $\#\mathrm{PM}(G) = 0$ then stop
   else nondet. choose between
         stop                                                    (* dummy additional path *)
         call GenTree($G$)
end.
procedure GenTree($G$: graph)
begin
   select an edge $e = (v, u)$ of $G$;
   $G_0 := G - \{u, v,$ and all edges incident to $v$ or $u\}$;
   $G_1 := G - \{e\}$;
   cases
      if both $\#\mathrm{PM}(G_i) > 0$ then nondet. choose between
         call GenTree($G_0$)
         call GenTree($G_1$);
      if for only one $i \in \{0, 1\}$ it holds $\#\mathrm{PM}(G_i) > 0$ then call GenTree($G_i$);
      if both $\#\mathrm{PM}(G_i) = 0$ then stop                (* $e$ is the only edge of $G$ *)
   end-cases
end
```

Showing that the computation tree of the above algorithm has $\#\mathrm{PM}(G) + 1$ leaves reduces to showing that for any graph G with *at least one perfect matching*, the computation tree of GenTree(G) has exactly $\#\mathrm{PM}(G)$ leaves. This can be shown by induction on the number of edges of G. Details are omitted. □

As an immediate corollary, it turns out that one of the central problems in Valiant's seminal paper on class #P [Val79a] is also TotP-complete under Cook [1]-reduction.

Corollary 3. *Computing the permanent of $n \times n$ $(0, 1)$-matrices is* TotP-*complete under Cook[1]-reduction.*

Remark 2. We can extend the above corollary to the problem of computing the permanent of any matrix with nonnegative integer entries. In Sect. 6 we will obtain this result as a consequence of the Main Theorem of Sect. 5.

5 TotP Is the Class of Self-reducible #PE Functions

5.1 Self-reducibility

In the previous section we proved that important #PE problems are in TotP. A key property of these problems is the possibility to reduce them to other instances of the same problem. Let us now formalize this notion of self-reducibility (somewhat different from Ko's self-reducibility [Ko83]):

Definition 2. *A function $f : \Sigma^* \to \mathbb{N}$ is called* poly-time self-reducible *if there exist polynomials r and q and polynomial time computable functions $h : \Sigma^* \times \mathbb{N} \to \Sigma^*$, $g : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, and $t : \Sigma^* \to \mathbb{N}$ such that for all $x \in \Sigma^*$:*

(a) $f(x) = t(x) + \sum_{i=0}^{r(|x|)} g(x,i) f(h(x,i))$, that is, f can be processed recursively by reducing x to $h(x,i)$, $(0 \leq i \leq r(|x|))$, and

(b) the recursion terminates after at most polynomial depth (that is, the value of f on instance $h\left(\ldots h(h(x,i_1),i_2)\ldots,i_{q(|x|)}\right)$ can be computed deterministically in polynomial time).

5.2 Main Theorem

We are now ready to state and prove the main theorem of this paper. Let $\#\mathsf{PE}_{\mathrm{SR}}$ be the class of all self-reducible functions of $\#\mathsf{PE}$.

Theorem 2. TotP *is exactly the closure under Karp reductions ($\leq^p_{\mathrm{m}}$) of* $\#\mathsf{PE}_{\mathrm{SR}}$.

Proof. 1. $\mathrm{Closure}_{\leq^p_{\mathrm{m}}}(\#\mathsf{PE}_{\mathrm{SR}}) \subseteq \mathsf{TotP}$:
It suffices to show that $\#\mathsf{PE}_{\mathrm{SR}} \subseteq \mathsf{TotP}$ because TotP is closed under $\leq^p_{\mathrm{m}}$.

Consider a function $f \in \#\mathsf{PE}_{\mathrm{SR}}$. Since $f \in \#\mathsf{PE}$, the question $f(x) > 0$ can be decided in polynomial time. This property, together with the existence of poly-time computable functions t, h, and g, and of polynomial r, such that $f(x) = t(x) + \sum_{i=0}^{r(|x|)} g(x,i) f(h(x,i))$, allows us to design a nondeterministic algorithm with a computation tree with exactly $f(x)+1$ leaves. A description of the algorithm follows.

```
Nondeterministic algorithm for a self-reducible #PE function $f$
(* On input $x$, the computation tree has $f(x)+1$ leaves *)
begin
  if $f(x) = 0$ then stop
  else nondet. choose between
      stop                                              (* dummy additional path *)
      call GenTree$_f(x)$
end.
procedure GenTree$_f(x)$
begin
  if $f(x)$ can be computed directly in polynomial time then
    spawn $f(x)$ nondeterministic branches
    at each branch stop                                 (* recursion has finished *)
  else
    compute $g(x,i), h(x,i)$, for all $i, 0 \leq i \leq r(|x|)$, and $t(x)$;
    check whether $g(x,i)f(h(x,i)) > 0, 0 \leq i \leq r(|x|)$, and whether $t(x) > 0$;
    nondet. choose between
      spawn $g(x,0)$ nondet. branches provided $g(x,0)f(h(x,0)) > 0$
        at each branch call GenTree$_f(h(x,0))$;            (* subtree with
                                                  $g(x,0)f(h(x,0))$ comp. paths *)
      ⋮
```

```
        spawn $g(x, r(|x|))$ nondet. branches provided $g(x, r(|x|))f(h(x, r(|x|))) > 0$
            at each branch call $\textsc{GenTree}_f(h(x, r(|x|)))$;                (* subtree with
                                                  $g(x, r(|x|))f(h(x, r(|x|)))$ comp. paths *)
        spawn $t(x)$ nondet. branches provided $t(x) > 0$
            at each branch stop;                          (* subtree with $t(x)$ comp. paths *)
end
(* The statement provided condition means that the corresponding nondeterministic
choice will be executed only if condition holds. *)
```

As in the proof of Theorem 1, it can be shown by induction on the recursion depth that the computation tree of $\textsc{GenTree}_f$ on input x has exactly $f(x)$ leaves.

Note that during the operation of $\textsc{GenTree}_f$ at least one of $g(x, i)f(h(x, i))$, $0 \leq i \leq r(|x|)$, $t(x)$ must be non-zero because $\textsc{GenTree}_f$ is called only for x's for which $f(x) > 0$. If only one of these values is nonzero, this step is actually deterministic.

By Condition (b) of Definition 2 the recursion depth is polynomial on $|x|$, hence each computation path needs polynomial time.

2. $\mathsf{TotP} \subseteq \mathrm{Closure}_{\leq^p_m}(\#\mathsf{PE}_{\mathrm{SR}})$:

Consider a TotP function f with corresponding PNTM M. Let M' be a modification of M which omits the leftmost computation path of M (if there are at least two). Clearly, for all x with $f(x) > 0$, it holds:

$$f(x) = \mathrm{tot}_M(x) = (\#\text{paths of } M \text{ on input } x) - 1 = \#\text{paths of } M' \text{ on input } x$$

Let us define a function $f' : \Sigma^* \times \Sigma^* \to \mathbb{N}$ associated with M' such that for every $x \in \Sigma^*$, $f'(x, y)$ gives the number of paths of the computation tree of M' on input x after following the nondeterministic choices described by y. If there is no such path, $f'(x, y) = 0$.

Claim: f Karp-reduces to f', i.e., $f \leq^p_m f'$. Indeed, let h be defined as follows:

$$h(x) = \begin{cases} (x, \varepsilon) & \text{if } f(x) > 0 \\ (x, 1^{q(|x|)+1}) & \text{otherwise} \end{cases}$$

where q is the polynomial that bounds the running time of M'.

Clearly, $h \in \mathsf{FP}$ and $f(x) = f'(h(x))$, which proves the claim.

We will now prove that $f' \in \#\mathsf{PE}_{\mathrm{SR}}$. It is not hard to see that $f' \in \mathsf{TotP}$ and hence, $f' \in \#\mathsf{PE}$. The following shows that f' is self-reducible:

- $f'(x, y) = f'(x, y0) + f'(x, y1) + t(x, y)$,
  where $t(x, y) = \begin{cases} 1 & \text{if } y \text{ corresponds to a leaf of } M'(x) \\ 0 & \text{otherwise} \end{cases}$
- $f'(x, y) = 0$ for $|y| = q(|x|) + 1$, hence the recursion depth is at most polynomial. □

6 Applications of the Main Theorem

Notice that the containment of #Perfect Matchings in TotP (Sect. 4) can in fact be obtained as an immediate corollary of the Main Theorem; actually one only needs the part $\mathrm{Closure}_{\leq^p_m}(\#\mathsf{PE}_{\mathrm{SR}}) \subseteq \mathsf{TotP}$. We will now show that several other #P-complete problems are in TotP and therefore are TotP-complete under Cook[1]-reduction.

More TotP-Complete Problems. The following problems are known (or can be easily shown) to be #P-complete [Val79a, Val79b, AJ90]:

1. #DNF-Sat: given a DNF Boolean formula, count the number of its satisfying assignments.
2. #Monotone-2-Sat: given a CNF Boolean formula with exactly two positive literals per clause, count the number of its satisfying assignments.
3. #NonCliques: given a graph G and number k, count the number of size-k subgraphs of G that *are not* cliques.
4. #NonIndependent Sets: given a graph G and number k, count the number of size-k subgraphs of G that are not independent sets.
5. NonNegative Permanent: given a $n \times n$ matrix A with nonnegative integer entries compute its permanent (cf. [Val79a]).
6. Ranking: given an NFA M and a word x, count the number of words $\leq x$ that are accepted by M.

Theorem 3. *The problems* #DNF-Sat*,* #Monotone-2-Sat*,* #NonCliques*,* #NonIndependent Sets*,* NonNegative Permanent*, and* Ranking *are* TotP*-complete under Cook[1] reduction.*

Proof. We prove the theorem for problem #NonCliques. Proofs for the remaining problems are omitted due to lack of space (to be included in the full version of the paper).

#NonCliques is #P-complete under Cook[1] reduction, because the problem of computing the number of size-k *cliques* of G which is #P-complete (under Karp, and therefore also under Cook[1] reduction) reduces to #NonCliques as follows:

$$\#\mathrm{Cliques}(G,k) = \binom{n}{k} - \#\mathrm{NonCl}(G,k)$$

Besides, it is easy to decide if one non-clique of size k exists: it happens so if and only if G is not a complete graph and $2 \leq k \leq n$ (n being the number of nodes of G). Therefore #NonCliques is in #PE. To prove containment in TotP we will show that #NonCliques is poly-time self-reducible.

Indeed, consider an instance of #NonCliques, that is a graph $G(V,E)$ and an integer k. Consider also a node v and let $u_1, \ldots, u_l$ be the nodes of G that are not adjacent to v. Let also:

$G_0 := (V \setminus \{v\}, E - \{\text{all edges incident to } v\})$ and
$G_1 := (V \setminus \{v, u_1, \ldots, u_l\}, E \setminus \{\text{all edges incident to } v, u_1, \ldots u_l\})$.

A careful observation reveals that the following are true:
$\#\mathrm{NonCl}(G_0, k) = \#$ non-cliques of size k that do not contain v
$\#\mathrm{NonCl}(G_1, k-1) = \#$ non-cliques of size k that contain v but none of the u_i's
$\sum_{i=2}^{l+1} \binom{n-i}{k-2} = \#$ non-cliques of size k that contain v and at least one of the u_i's.
Therefore:

$$\#\mathrm{NonCl}(G,k) = \#\mathrm{NonCl}(G_0,k) + \#\mathrm{NonCl}(G_1,k-1) + \sum_{i=2}^{l+1} \binom{n-i}{k-2}$$

For correctness, we also need to define that for any G, $\#\mathrm{NonCl}(G, 1) = 0$ and $\#\mathrm{NonCl}(G, k) = 0$ if $k > n$.

The above equality implies that #NonCliques has the self-reducibility property and therefore belongs to TotP. Hence, #NonCliques is TotP-complete under Cook[1] reduction. □

7 TotP-Completeness Under Many-One Reductions

As already mentioned, all #P-complete problems under the Cook[1] reduction that belong to TotP (e.g., #Perfect Matchings, #DNF-Sat, #NonCliques) are also TotP-complete (and #PE-complete) under Cook[1]; and vice-versa, since #P Cook[1]-reduces to TotP. We will now focus to TotP-completeness in the many-one sense.

TotP is a 'syntactic' class (each PNTM defines a function in TotP); hence it has a complete function under the Karp reduction. We would like to know whether there are natural TotP-complete problems under some many-one reduction[3]. We do not know the answer yet but there are some *hints*. The first is that any such problem has to be TotP-complete under the corresponding 1-Turing reduction (presumably Cook[1] or some stronger reduction, e.g., logspace 1-Turing), and therefore also #P-complete under the Cook[1] reduction, since #P Cook[1]-reduces to TotP. The second hint is given by the following proposition, which is an immediate consequence of the fact that many-one reductions between functions preserve the function value.

Proposition 2. *If a function f that admits an FPRAS is many-one complete for* TotP *then all functions in* TotP *admit an FPRAS.*

(*FPRAS* stands for "Fully Polynomial Randomized Approximation Scheme".)

Remark 3. The above implies that if #Perfect Matchings or #DNF-Sat is TotP-complete under some many-one reduction, then all functions in TotP admit an *FPRAS* since these two problems admit an *FPRAS*, as was shown in

[3] Here we mean not only the Karp reducibility (polynomial-time many-one) but any reasonable many-one reducibility, e.g., the logarithmic space one.

[JS96] and [KLM89] respectively. This in turn would imply that problem #BIS (counting independent sets in a bipartite graph) as well as several problems which are AP-interreducible to #BIS [DGGJ03] also admit an *FPRAS*, because it can be easily seen that #BIS belongs to TotP; however, it was noted in [DGGJ03] that no *FPRAS* is known for any of these problems.

8 Conclusions

In this paper we have investigated the complexity of counting functions with easy decision version, with the help of two recently introduced complexity classes: #PE, which contains all these functions, and TotP, which contains functions that count the *total* number of paths of PNTMs.

Our main result is that TotP is exactly the Karp-closure of self-reducible #PE functions. Important counting problems such as #Perfect Matchings, #DNF-Sat, NonNegative Permanent, and #NonCliques, belong to this class. This implies that they all possess a new computational model, namely for each of them there is a PNTM with as many computation paths as the function value. This characterization may help discover new properties of such problems.

On the other hand, it could be the case that these problems belong to an even smaller class, e.g., a subclass of TotP. Hence, we would like to be able to determine their exact complexity. That is, to show that #Perfect Matchings (or #DNF-Sat, or #NonCliques, etc.) is TotP-complete under Karp reduction, or find a subclass of TotP, closed under Karp reduction, for which these problems are Karp-complete. The latter seems to be more likely, taking into account the remark that follows Proposition 2. Another promising approach, proposed in [DHK00], is to employ other (weaker) kinds of reductions under which #P, #PE, and TotP are closed.

Finally, let us remark that subclasses of #P that contain functions with easy decision version have been defined in [SST95] and [HHKW05]. It is an interesting open question to see how these classes relate to TotP and #PE.

Acknowledgments. We would like to thank Jacobo Torán, Johannes Köbler, Phokion Kolaitis, and Mihalis Yannakakis for discussions and observations that helped us clarify notions and improve the readability of the paper.

References

[AJ90] C. Àlvarez, B. Jenner. A very hard log space counting class. In *Proceedings of Structure in Complexity Theory Conference 1990*: 154–168.

[DHK00] A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science* 340(3): 496-513, 2005.

[DGGJ03] M. Dyer, L. A. Goldberg, C. Greenhill, and M. Jerrum. On the relative complexity of approximate counting problems. *Algorithmica* 38(3): 471–500 (2003).

[HHKW05] L.A. Hemaspaandra, C.M. Homan, S. Kosub, and K.W. Wagner. The complexity of computing the size of an interval. Technical Report cs.cc/0502058, ACM Computing Research Repository, February 2005. (Preliminary version: L.A. Hemaspaandra, S. Kosub, and K.W. Wagner, The complexity of computing the size of an interval, in Proceedings ICALP 2001, LNCS 2076, pp. 1040-1051, Springer 2001.)

[JS96] M. Jerrum and A. Sinclair. The Markov chain Monte-Carlo method: an approach to approximate counting and integration. In *Approximation Algorithms for NP-hard Problems* (Dorit Hochbaum, ed.), PWS, pp. 482–520, 1996.

[KLM89] R.M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms* 10: 429–448 (1989).

[KPSZ98] A. Kiayias, A. Pagourtzis, K. Sharma, and S. Zachos. The complexity of determining the order of solutions. In *Proceedings of the First Southern Symposium on Computing*, Hattiesburg, Mississippi, 4–5 December 1998.

[KPZ99] A. Kiayias, A. Pagourtzis, and S. Zachos. Cook Reductions Blur Structural Differences Between Functional Complexity Classes. In *Proceedings of the 2nd Panhellenic Logic Symposium*, pp. 132-137, Delphi, 13–17 July 1999.

[Ko83] On Self-Reducibility and Weak P-Selectivity. *Journal of Computer and System Sciences*, 26(2):209–221, 1983.

[Kre88] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, June 1988.

[KST89] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26(4):363–379, 1989.

[Pag01] A. Pagourtzis. On the complexity of hard counting problems with easy decision version. In *Proceedings of the 3rd Panhellenic Logic Symposium*, Anogia, Crete, 17–21 July 2001.

[SST95] S. Saluja, K. V. Subrahmanyam, and M. N. Thakur. Descriptive Complexity of #P Functions. *Journal of Computer and System Sciences*, 50(3): 493–505 (1995).

[Sim75] J. Simon. *On some Central Problems of Computational Complexity*. PhD thesis, Cornell University, Ithaca, NY, 1975.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[TW92] S. Toda and O. Watanabe. Polynomial-time 1-Turing reductions from ♯PH to ♯P. *Theoretical Computer Science*, 100(1):205–221, 1992.

[Tor88] J. Toran. *Structural Properties of the Counting Hierarchies*. PhD thesis, Facultat d'Informatica de Barcelona, 1988.

[Val79a] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, April 1979.

[Val79b] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, August 1979.

[Vol94] H. Vollmer. On different reducibility notions for function classes. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lecture Notes in Computer Science*, pages 449–460, Caen, France, 24–26 February 1994. Springer.

[Wag86] K. W. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47(2):131–147, 1986.

On Non-Interactive Zero-Knowledge Proofs of Knowledge in the Shared Random String Model

Giuseppe Persiano and Ivan Visconti

Dipartimento di Informatica ed Appl., Università di Salerno, Italy
{giuper, visconti}@dia.unisa.it

Abstract. In this paper we study the notion of a *Double-Round* NIZKPK in the SRS model. In a Double-Round NIZKPK prover and verifier have access to the same random string Σ and, in addition, the prover is allowed to send one message to the verifier before Σ is made available. The verifier needs not to reply to this message. The random string and initial prover message can then be used in any polynomial number of proofs each consisting of a single message.

We show how to construct Double-Round *non-malleable* NIZKPKs in the SRS model by only requiring the existence of *one-way trapdoor permutations*. In contrast, regular NIZKPKs require the existence of cryptosystems with an extra density property, called dense secure cryptosystems. We then show that Double-Round NIZKPKs can replace one-round NIZKPKs in the design of secure protocols. The replacement has no significant effect on the round complexity of the larger protocol but it removes the need of the existence of dense secure cryptosystems. We give examples of cryptographic constructions that use one-round NIZKPKs and that are improved when using Double-Round NIZKPKs: 1) the construction of 3-round resettable zero-knowledge arguments in the UPK model [EUROCRYPT 2001]; 2) the construction of a constant-round $(n-1)$-secure simulatable coin-flipping protocol [EUROCRYPT 2003].

1 Introduction

Non-Interactive Zero Knowledge (NIZK, for short) is an important cryptographic primitive originally shown to exist in the *shared random string* (SRS, for short) model [5,11] where the prover and the verifier share a random string and the prover gives one-message zero-knowledge proofs.

Besides its conceptual significance within the theory of zero knowledge, NIZK has been an important building block for several cryptographic protocols. For instance, simulation-sound NIZK is used in the constructions of public-key cryptosystems secure under chosen ciphertext attacks [21] and in the constructions of group signature schemes based on general complexity assumptions [3]. The notion of a NIZK proof of knowledge (NIZKPK, for short) has been used for constructing non-malleable [9] NIZKPKs [6], non-malleable interactive zero-knowledge arguments [1], round-efficient resettable zero-knowledge arguments in the UPK model [18] and $(n-1)$-secure simulatable coin flipping protocols [15].

All these constructions work in the SRS model and consider one round per proof sent by the prover to the verifier. Recently in [2] a non-malleable NIZKPK has been constructed in a specific trusted PKI model (which is a relaxed assumption compared to the CRS model) by only requiring the existence of one-way trapdoor permutations. Under the same assumptions, the same result can be achieved in the common reference string model using specific reference strings, but unfortunately in this case the reference string must be generated by a trusted third party (i.e., it can not be taken from a natural phenomenon or as result of a coin-flipping protocol).

An important issue is the design of NIZK proofs of membership and proofs of knowledge based on as weak as possible assumptions. In this paper we focus on computational zero knowledge and for one-round NIZK proofs of membership in the SRS model, the weakest assumption known to be sufficient is the existence of one-way trapdoor permutations [11,4] (we only consider proof system with efficient provers) whereas for one-round NIZK proofs of knowledge (NIZKPK, for short) so-called dense secure cryptosystems in addition to one-way trapdoor permutations have been shown to be sufficient [8] and necessary [7] (for non-trivial relations). This state of things contrasts with the interactive case in which one-way functions are known to be sufficient [14] and necessary [19] (for non-trivial languages) and to the CRS and the specific trusted PKI models where dense secure cryptosystems are not necessary for the construction of one-round NIZKPKs.

We stress that the SRS model assumes that prover and verifier have access to the same uniformly distributed random string, and thus it is assumed that there exists a source of randomness (e.g., the result of a natural event or a trusted third party that works as dealer) that is not under the control of the parties. If instead such a source of randomness does not exist, the standard way of using NIZK as sub-protocol of a larger interactive protocol involves running a coin-tossing protocol to construct a random string Σ and then use Σ as the shared random string for NIZK proofs. Such an approach is often useful to improve the round-complexity of protocols [18,10] or to exploits some nice properties of NIZK proofs [3,21,1].

1.1 New Constructions and Results

In this paper, we define *Double-Round* NIZK proofs in the SRS model. Here, similarly to one-round NIZK proofs in the SRS model of [5,11], prover and verifier have access to the same random string Σ and, in addition, we allow the prover to send one message to the verifier before Σ is made available. The verifier needs not to reply to this message. The same random string and initial prover message can then be used by the prover to prove any polynomial number of theorems using a single message for each of them.

We show how to construct Double-Round NMNIZKPKs under the sole assumption that one-way trapdoor permutations exist. In contrast, in the SRS model one-round NIZKPKs for non trivial relations are known to require dense secure cryptosystems [7] (even if non-malleability is not required) and can be constructed using one-way trapdoor permutations and dense secure cryptosystems (by using the NIZK proof of membership by [11] as a building block in the construction of NIZKPK of [8] which in turn can be used to obtain non-malleable NIZKPK as shown in [6]).

More precisely, our results are the following.

1. In Theorem 1, we show that, assuming existence of *one-way trapdoor permutations*, there exists a Double-Round *single-theorem* NIZKPK Δ in the SRS model for all polynomial-time relations. The main technique used for this result is the combined use of two different extractable commitments.
2. We show that Δ is a Double-Round *unbounded* Non-Interactive *Witness-Indistinguishable* Proof of Knowledge (NIWIPK, for short). We stress that this second step towards our construction of Double-Round non-malleable NIZKPK requires some attention. Indeed, unlike one-round NIZK in the SRS model, not all Double-Round single-theorem NIZKPKs are also unbounded witness indistinguishable.
3. We prove the existence of Double-Round unbounded NMNIZKPKs in the SRS model under the assumption that one-way trapdoor permutations exist. The result is obtained by following the transformation of [6] that constructs an unbounded NMNIZKPK in the SRS model.

Although Double-Round NIZK in the SRS model are weaker than (one-round) NIZK in the SRS model, Double-Round NIZK proofs retains two important properties of NIZK proofs in the SRS model. First of all, communication is still mono-directional: from prover to verifier only. Thus the zero-knowledge and the extractability properties are trivially preserved if proofs are concurrently composed (after the shared random string has been made available). Moreover, we insist on the shared string Σ to be uniformly distributed. Thus, Σ can be obtained as the result of probabilistic natural phenomenon. If we remove this requirement, then we obtain the so-called CRS model which is more difficult to justify.

We finally stress that one can see an implicit interaction required to guarantee that the first prover message is sent before the SRS is announced. However, this holds in part for the classical notion of one-round NIZK since the proofs can not be sent before the SRS is announced. The concrete notion of non-interactiveness in both cases concerns the fact that the announcement of the first prover message (in the double-round case) and the announcement of the SRS (in both cases) has to be done only once.

Applications to Protocol Design. Double-Round NIZKPK can replace one-round NIZKPK in the design of secure protocols. The replacement does not significantly increase the number of rounds of the protocol and it reduces the assumptions needed from existence of dense secure cryptosystems *and* one-way trapdoor permutations (which are currently the weakest assumptions known to suffice for one-round NIZKPK [11,8]) to the assumption of existence of one-way trapdoor permutations (which we show in this paper to be sufficient for Double-Round NIZKPK). We stress also that, by the results of [7], dense secure cryptosystems are necessary for one-round NIZKPK in the SRS model, therefore any construction based on one-round NIZKPK will need to assume the existence of dense secure cryptosystems.

We exemplify the power of Double-Round NIZKPK in the SRS model by discussing applications to constructions of secure protocols. In particular in [20], we show ho to use Double-Round NIZKPK for removing the need of dense secure cryptosystem from the 3-round concurrently sound resettable zero-knowledge argument system of [18] and from the $(n-1)$-secure simulatable coin flipping protocol of [15].

Remark on certified *One-Way Trapdoor Permutations.* In the rest of the paper we will consider *certified* one-way trapdoor permutations. These permutations admit an efficient algorithm that verifies whether the description of a function f corresponds to a permutation. An implementation of such a primitive can be based for instance on the RSA assumption (see [16]).

2 Double-Round NIZK Proofs

Definition 1. *$\Delta = (\ell, \mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V})$ is a* Double-Round non-interactive proof system *for the language $L \in \mathcal{NP}$ if:*

1. *ℓ is a polynomial, $\mathcal{P}_1, \mathcal{P}_2, \mathcal{V}$ are probabilistic polynomial-time algorithms.*
2. Completeness*: for all $x \in L$ of length k, for all y such that $R_L(x,y) = \texttt{true}$, for all strings Σ of length $\ell(k)$, we have that*

$$\text{Prob}\left[\,(\texttt{Pre}, \texttt{PreAux}) \leftarrow \mathcal{P}(1^k); \mathcal{V}(x, \texttt{Pre}, \mathcal{P}_2(x, y, \texttt{Pre}, \Sigma, \texttt{PreAux}), \Sigma) = \texttt{true}\right] = 1.$$

3. Soundness*: for all pairs of algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have that*

$$\text{Prob}\left[(\texttt{Pre}, \texttt{PreAux}) \leftarrow \mathcal{A}_1(1^k);\ \Sigma \leftarrow \{0,1\}^{\ell(k)}; (x, \Pi) \leftarrow \mathcal{A}_2(\texttt{Pre}, \Sigma, \texttt{PreAux}) : \\ x \notin L \text{ and} \mathcal{V}(x, \texttt{Pre}, \Pi, \Sigma) = \texttt{true}\ \right] < \nu(k),$$

for some negligible function ν.

Definition 2. *$\Delta = (\ell, \mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V}, \mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2))$ is a Double-Round* single-theorem *NIZK proof system for the language L if:*

1. *$(\ell, \mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V},)$ is a Double-Round non-interactive proof system for L*
2. *$\mathcal{S}_1, \mathcal{S}_2$ are probabilistic polynomial-time algorithms;*
3. *Single-Theorem Zero Knowledge: for all non-uniform probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have that*

$$\left|\text{Prob}\left[\,\texttt{Expt}_{\mathcal{A}}(k) = 1\,\right] - \text{Prob}\left[\,\texttt{Expt}^{\mathcal{S}}_{\mathcal{A}}(k) = 1\,\right]\right| < \nu(k)$$

for some negligible function ν, where the experiments $\texttt{Expt}_{\mathcal{A}}(k)$ and $\texttt{Expt}^{\mathcal{S}}_{\mathcal{A}}(k)$ are defined as follows:

```
$\texttt{Expt}_{\mathcal{A}}(k)$:
    $(\texttt{Pre}, \texttt{PreAux}) \leftarrow \mathcal{P}_1(1^k)$
    $\Sigma \leftarrow \{0,1\}^{\ell(k)}$
    $(x, y, s) \leftarrow \mathcal{A}_1(\texttt{Pre}, \Sigma) : (x, y) \in R_L$
    $\Pi \leftarrow \mathcal{P}_2(x, y, \texttt{Pre}, \Sigma, \texttt{PreAux})$
    **return** $\mathcal{A}_2(\texttt{Pre}, \Pi, s)$
```

```
$\texttt{Expt}^{\mathcal{S}}_{\mathcal{A}}(k)$ :
    $(\texttt{Pre}, \Sigma, \texttt{PreAux}) \leftarrow \mathcal{S}_1(1^k)$
    $(x, y, s) \leftarrow \mathcal{A}_1(\texttt{Pre}, \Sigma) : (x, y) \in R_L$
    $\Pi \leftarrow \mathcal{S}'(x, y, \texttt{Pre}, \Sigma, \texttt{PreAux})$
    **return** $\mathcal{A}_2(\texttt{Pre}, \Pi, s)$
```

where $\mathcal{S}'(x, y, \texttt{Pre}, \Sigma, \texttt{PreAux}) \stackrel{\text{def}}{=} \mathcal{S}_2(x, \texttt{Pre}, \Sigma, \texttt{PreAux})$.

In a Double-Round single-theorem NIZK proof system the preprocessing message of the prover can be used to give only one non-interactive zero-knowledge proof. We stress though that the zero-knowledge property is guaranteed even if the statement is adversarially chosen after the preprocessing stage has been performed and the string Σ has appeared. In a Double-Round unbounded NIZK proof system instead the simulator is required to simulate a polynomial number of proofs on adversarially and adaptively chosen instances. In other words, the same first message Pre and the same random string Σ can be used to give any polynomial number of proofs.

Definition 3. *A Double-Round single-theorem NIZK proof system for the language* $L \in \mathcal{NP}$ $\Delta = (\ell, \mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V}, \mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2))$ *is a* Double-Round unbounded NIZK proof system *for* L *if the following property holds.*
Unbounded Zero Knowledge*: for all non-uniform probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* ν *such that, for all sufficiently large* k*, we have that*

$$\left|\text{Prob}\left[\texttt{Expt}_{\mathcal{A}}(k) = 1\right] - \text{Prob}\left[\texttt{Expt}^{\mathcal{S}}_{\mathcal{A}}(k) = 1\right]\right| < \nu(k)$$

where the experiments $\texttt{Expt}_{\mathcal{A}}(k)$ *and* $\texttt{Expt}^{\mathcal{S}}_{\mathcal{A}}(k)$ *are defined as follows:*

```
$\texttt{Expt}_{\mathcal{A}}(k)$ :
    $(\texttt{Pre}, \texttt{PreAux}) \leftarrow \mathcal{P}_1(1^k)$
    $\Sigma \leftarrow \{0,1\}^{\ell(k)}$
    **return** $\mathcal{A}^{\mathcal{P}_2(\cdot,\cdot,\texttt{Pre},\Sigma,\texttt{PreAux})}(\Sigma)$
```

```
$\texttt{Expt}^{\mathcal{S}}_{\mathcal{A}}(k)$ :
    $(\texttt{Pre}, \Sigma, \texttt{PreAux}) \leftarrow \mathcal{S}_1(1^k)$
    **return** $\mathcal{A}^{\mathcal{S}'(\cdot,\cdot,\texttt{Pre},\Sigma,\texttt{PreAux})}(\Sigma)$
```

where $\mathcal{S}'(x, y, \texttt{Pre}, \Sigma, \texttt{PreAux}) \stackrel{\text{def}}{=} \mathcal{S}_2(x, \texttt{Pre}, \Sigma, \texttt{PreAux})$ *and the pair of the first two inputs to* $\mathcal{P}_2$ *and* $\mathcal{S}'$ *is in* R_L.

Next we present the notion of Double-Round non-interactive proof of *knowledge*. We follow the standard notion of extraction in the SRS model (see [8]) by considering straight-line extraction only. In order to achieve witness extraction, the extractor algorithm that establishes the random string will run as input the first round of the adversarial prover.

Definition 4. $\Delta = (\ell, \mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2), \mathcal{V}, \mathrm{E} = (\mathrm{E}_1, \mathrm{E}_2))$ *is a Double-Round non-interactive* proof of knowledge *(NIPK, for short) for the language $L \in \mathcal{NP}$ with witness relation R_L if:*

1. $(\ell, \mathcal{P}, \mathcal{V})$ *is a Double-Round non-interactive proof system for L;*
2. $(\mathrm{E}_1, \mathrm{E}_2)$ *is a pair of probabilistic polynomial-time algorithms such that for any pair of algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and for sufficiently large k there exists a negligible function $\nu(k)$ such that*

$$\mathrm{Prob}\left[\mathtt{Expt}_{\mathcal{A}}^{\mathrm{E}}(k)\right] \geq \mathrm{Prob}\left[\mathtt{Expt}_{\mathcal{A}}(k)\right] - \nu(k),$$

where the experiments $\mathtt{Expt}_{\mathcal{A}}(k)$ and $\mathtt{Expt}_{\mathcal{A}}^{\mathrm{E}}(k)$ are defined as follows:

$\mathtt{Expt}_{\mathcal{A}}(k)$:	$\mathtt{Expt}_{\mathcal{A}}^{\mathrm{E}}(k)$:
$(\mathtt{Pre}, \mathtt{PreAux}) \leftarrow \mathcal{A}(1^k)$	$(\mathtt{Pre}, \mathtt{PreAux}) \leftarrow \mathcal{A}(1^k)$
$\Sigma \leftarrow \{0,1\}^{\ell(k)}$	$(\Sigma, \mathtt{aux}) \leftarrow \mathrm{E}_1(\mathtt{Pre})$
$(x, \Pi) \leftarrow \mathcal{A}(\mathtt{Pre}, \Sigma, \mathtt{PreAux})$	$(x, \Pi) \leftarrow \mathcal{A}(\mathtt{Pre}, \Sigma, \mathtt{PreAux})$
return $\mathcal{V}(x, \mathtt{Pre}, \Pi, \Sigma)$	$y \leftarrow \mathrm{E}_2(\Sigma, \mathtt{aux}, x, \mathtt{Pre}, \Pi)$
	return $\mathtt{true}$ *if* $(x, y) \in R_L$

Note that the existence of such a pair of algorithms (E_1, E_2) referred to as witness extraction *implies the soundness condition.*

The definitions of Double-Round *non-malleable* NIZK proof system, Double-Round *simulation sound* NIZK proof system and Double-Round witness indistinguishable non-interactive proof system can be found in the full version of the paper [20].

3 Double-Round NIZK Proofs of Knowledge

In this section we show a Double-Round single-theorem NIZKPK in the SRS model that is based on the existence of certified one-way trapdoor permutations. Here, we consider a generic $\mathcal{NP}$-language L and assume that for a string $x \in L$ of length n, there exists a witness of length at most $m = \mathtt{poly}(n)$.

3.1 Assumptions and Tools

Our constructions are based on the following tools.

Certified One-Way Trapdoor Permutations. We assume the existence of a family $\mathcal{F}$ of *certified* one-way trapdoor permutations. Specifically, we have a quadruple of efficient algorithms (GenT, EvalT, InvT, VerT). The *generator algorithm* GenT, on input a security parameter 1^k, outputs the description of a permutation f over $\{0,1\}^k$ and the associated trapdoor information t_f. We will assume wlog that the size of t_f is k. The *evaluation algorithm* $\textsc{EvalT}$, on input the description of a permutation f over $\{0,1\}^k$ and a string $x \in \{0,1\}^k$, outputs $f(x)$. The *inverting algorithm* $\textsc{InvT}$, on input the description of permutation f over $\{0,1\}^k$, the associate trapdoor information t_f and $z \in \{0,1\}^k$, outputs $f^{-1}(z)$. The *verifying algorithm* $\textsc{VerT}$ verifies whether the description of a function f corresponds to a permutation of $\mathcal{F}$. Such a primitive can be implemented for instance by using the RSA assumption (see [16]).

Non-Interactive Extractable Commitment Schemes. Intuitively a commitment scheme can be seen as the digital equivalent of a sealed envelope. If a party A wants to commit to some message m she just puts it into the sealed envelope, so that whenever A wants to reveal the message, she opens the envelope. Clearly, such a mechanism can be useful only if it meets some basic requirements. First of all the digital envelope should *hide* the message: no party other than A should be able to learn m from the commitment (this is often referred in the literature as the hiding property). Second, the digital envelope should be *binding*, meaning with this that A can not change her mind about m, and by checking the opening of the commitment one can verify that the obtained value is actually the one A had in mind originally (this is often referred to as the binding property).

Informally, a non-interactive extractable commitment scheme is a non-interactive commitment scheme with an efficient extractor that, on input a commitment `com` of a message m and a special trapdoor, outputs m. The following two main variants of extractable commitment schemes are known. We denote by `HC` a hard-core predicate for a family of certified one-way trapdoor permutations $\mathcal{F}$ [13].

1. **Non-Interactive Extractable Commitment in the CRS Model**: In this case the extractor receives as auxiliary input a trapdoor corresponding to the information stored in the reference string; such commitment schemes exist for instance under the assumption that secure cryptosystems exist [17].
2. **Non-Interactive Extractable Commitment in the SRS Model**: In this case the extractor receives as auxiliary input a trapdoor corresponding to the shared random string; such commitment schemes have been shown to exist if and only if dense secure cryptosystems exist [7].

A main tool that we construct and use in our constructions of Double-Round NIZKPK is a *non-interactive extractable commitment scheme* in the SRS model with the following additional requirements:

1. the committer algorithm sends a set-up message before the shared random string appears;
2. once the shared random string is available, the committer algorithm can output any polynomial number of non-interactive extractable commitments.

Our construction of Double-Round NIZKPK implies the existence of such commitment schemes under the assumptions that certified one-way trapdoor permutations exist; we stress that both commitment schemes known in literature and discussed above require either a common reference string or the existence of dense secure cryptosystems. Therefore, the resulting construction for an extractable commitment scheme is novel and of independent interest.

In our construction of Double-Round NIZKPK we will use the following two extractable commitment schemes that exist under the assumption that one-way certified trapdoor permutations exist.

Deterministic Extractable Commitment in the SRS Model. Let f be a permutation over $\{0,1\}^k$ (for which the associated trapdoor t_f is *known* to the committer) and let Σ be a k-bit shared random string. Then to commit to a bit b it is enough to compute $\texttt{com} = \texttt{HC}(f^{-1}(\Sigma)) \oplus b$. The "commitment" key is simply $\texttt{com}$. It is easy to see that the commitment is perfectly binding (f is a permutation) and computationally hiding (f is one-way). We will denote by $\textsc{Com}(\Sigma, (f, t_f), b)$ the algorithm that returns a pair $(\texttt{com}, \texttt{dec})$ of commitment/decommitment keys of bit b with shared random string Σ, certified one-way trapdoor permutation f and corresponding trapdoor t_f. We observe that if $f^{-1}(\Sigma)$ is known, it is possible to extract the bit committed to by $\texttt{com}$, i.e., the commitment scheme described above is an extractable commitment scheme in the SRS model. The notation is easily extended to $\textsc{Com}(\Sigma, (f, t_f), w)$ with w of length l and Σ of length lk. Note that once the shared random string Σ is established, the above described commitment scheme is deterministic and therefore if two different strings are committed by using Σ then the hiding property does not hold anymore.

Extractable Commitment in the CRS Model. In a different scheme, for committing to a bit b using as reference string a certified one-way trapdoor permutation g, the committer picks a random $z \in \{0,1\}^k$ such that $\texttt{HC}(z) = b$ and computes the commitment $\texttt{com} = g(z)$. The commitment scheme can be easily extended to strings and, with a slight abuse of notation, we denote by $\textsc{Com}(g, w)$ the algorithms that returns a pair $(\texttt{com}, \texttt{dec})$ of commitment/decommitment keys for the string w using certified one-way trapdoor permutation g. The decommitment key $\texttt{dec}$ associated with commitment $\texttt{com} = (\texttt{com}_1, \ldots, \texttt{com}_l)$ is simply the sequence $z_1, \ldots, z_l$ where, $g(z_i) = \texttt{com}_i$ for $i = 1, \ldots, l$ and $w = \texttt{HC}(z_1) \cdots \texttt{HC}(z_l)$. We make the two simple observations that if the trapdoor information associated with g is known, then it is possible to extract the value committed to by $\texttt{com}$ and that knowledge of the trapdoor information is not needed to compute the commitment. Moreover, the same certified one-way trapdoor permutation g (i.e., the same shared random string) can be used to perform any polynomial number of commitments while still preserving the hiding property.

One-Round NIZK Proofs. We denote by $(\textsc{Prover}, \textsc{Verifier})$ a one-round NIZK proof system for an $\mathcal{NP}$-complete language, say the language HAM of Hamiltonian graphs. The one-round NIZK proof system $(\textsc{Prover}, \textsc{Verifier})$ can be constructed under the assumption of the existence of certified one-way trapdoor permutations [11] and we denote by $\mathcal{S}^H = (\mathcal{S}_1^H, \mathcal{S}_2^H)$ the associated simulator. Algorithm $\textsc{Prover}$ takes three arguments: an instance x, a random string Σ, and a witness y for x. The simulator $\mathcal{S}^H = (\mathcal{S}_1^H, \mathcal{S}_2^H)$ works into two stages: first it generates a string Σ along with auxiliary information $\texttt{aux}$; then, on input x, the string Σ, and the auxiliary information $\texttt{aux}$, it outputs a "proof" Π.

We will use the two following languages $\Lambda_0, \Lambda_1 \in \mathcal{NP}$:

1. $\tau_0 = (f, g, \Sigma, \texttt{comTrap}) \in \Lambda_0$ if `comTrap` is an extractable commitment of the trapdoor information t_g with respect to g in the SRS model; Σ is the shared random string and f is the certified one-way trapdoor permutation used in the commitment scheme.
2. $\tau_1 = (x, g, \texttt{comWit}) \in \Lambda_1$ if `comWit` is an extractable commitment of a witness y for "$x \in L$" computed by means of the permutation g in the CRS model.

In the description of our proof systems we will often run Prover, Verifier, $\mathcal{S}^H$ of a one-round NIZK proof system for the $\mathcal{NP}$-complete language HAM on input an instance of Λ_0 (resp. Λ_1) and we actually mean that the standard reduction from Λ_0 (resp. Λ_1) to an instance for Hamiltonicity is applied before the one-round NIZK proof system starts.

Deterministic One-Round NIZK Proof Systems. In general, in a one-round NIZK proof system the prover is assumed to be randomized and thus, for the same shared random string and the same theorem, the prover might give different proofs. However, if we fix the random tape of the prover then the prover becomes deterministic and the same proof is produced with respect to the same shared random string and the same theorem. We notice though that such a deterministic prover might lose the zero-knowledge property should the same random tape be used for the same reference string and different theorems. However, as it will be evident in what follows, we will use a deterministic prover only to prove the *same* theorem several times with respect to the *same* shared random string.

We denote by Prover$(R; \ldots)$ and by $\mathcal{S}(R; \ldots)$ the execution of a prover Prover$(\ldots)$ or a simulator $\mathcal{S}(\ldots)$ for a one-round NIZK proof system when R is used by the prover and by the simulator as random tape.

3.2 Construction for Double-Round NIZKPK

We now discuss our construction of Double-Round NIZKPK based on the existence of any certified one-way trapdoor permutation. In order to achieve the result, we crucially use the deterministic one-round NIZK proof systems and the extractable commitment schemes in the CRS and in the SRS models discussed above. In particular we obtain a Double-Round unbounded NIWI proof system that then can be used to obtain a Double-Round unbounded non-malleable NIZKPK by means of the transformation of [6].

Overview. In the preprocessing stage, the prover picks a random tape R and two certified one-way trapdoor permutations f, g along with their trapdoors t_f and t_g and sends (f, g) to the verifier. Once the shared random string $\Sigma = \Sigma_1 \circ \Sigma_2 \circ \Sigma_3$ is made available the prover performs the following steps to produce a proof Π that he knows a witness y for $x \in L$.

1. The shared random string Σ_1 is used to compute the commitment `comTrap` of the trapdoor t_g by means of (f, t_f) and using the extractable commitment scheme in the SRS model discussed above. Notice that this commitment is deterministic (for fixed Σ_1) and all proofs Π produced by the prover will contain the same commitment `comTrap` of t_g.

2. The permutation g is used to compute the pair $(\mathtt{comWit}, \mathtt{decWit})$ of commitment and decommitment keys of the witness y for "$x \in L$" by using the extractable commitment scheme in the CRS model discussed above. Note that the commitments comWit will differ from proof to proof.
3. The random string Σ_2 and knowledge of decTrap are used to compute a one-round NIZK proof (of membership) π_0 that $\tau_0 = (f, g, \Sigma_1, \mathtt{comTrap}) \in \Lambda_0$. Notice that τ_0 is the same for all proofs Π that the prover will produce and, since the prover uses the same random tape R, all proofs π will share the same proof π_0 that $\tau_0 \in \Lambda_0$.
4. Finally, the random string Σ_3 and knowledge of $(y, \mathtt{decWit})$ are used to compute a one-round NIZK proof (of membership) π_1 that $\tau_1 = (x, g, \mathtt{comWit}) \in \Lambda_1$.

A formal description of the protocol is found in Figure 1.

The zero-knowledge property follows from the hiding properties of the commitments and the zero-knowledge of the one-round NIZK proof system employed. For the proof of knowledge property, the idea is to construct the string $\Sigma_1 = (\Sigma_{1,1}, \cdots, \Sigma_{1,k})$ by picking z_i at random from $\{0,1\}^k$ and setting $\Sigma_{1,i} = f(z_i)$, where f is the one-way trapdoor permutation chosen by the prover in the first message. Notice that since f is a permutation the string Σ_1 is uniformly distributed. Thus when the prover produces comTrap the extractor manages to extract the trapdoor t_g of g. Knowledge of t_g then allows the extractor to obtain the witness y from comWit.

Theorem 1. *Under the assumption that certified one-way trapdoor permutations exist, there exists (constructively) a single-theorem Double-Round NIZK proof of knowledge for $\mathcal{NP}$.*

See [20] for the proof.

Double-Round NIWI and Non-Malleable Proofs. Our goal is to design a Double-Round non-malleable unbounded NIZKPK and one could (incorrectly) think that the transformation of [6] that constructs a one-round NMNIZKPK from a (possibly malleable) one-round single-theorem NIZKPK could be used. Then, the results should follow by applying the transformation of [6] to the Double-Round single-theorem NIZKPK of the previous section. We observe though that the transformation of [6] implicitly uses the following properties of one-round NIZK in the SRS model (proved in [12,11]): 1) any one-round single-theorem NIZK proof system is also a one-round single-theorem NIWI proof system; 2) any one-round single-theorem NIWI proof system is also a one-round unbounded NIWI proof system.

However consider the simplified Double-Round single-theorem NIZKPK proof system in which only one certified one-way trapdoor permutation f is used and the witness y for "$x \in L$" is committed to by using only the extractable commitment in the SRS model. As we discussed above, such a commitment scheme is deterministic (for a fixed shared random string Σ) and thus, if the same shared random string Σ is used to commit to more than one witness, then the

```
Preprocessing Stage
    ALGORITHM $\mathcal{P}_1(1^k)$
     1. randomly pick a random tape $R$ and certified one-way trapdoor permutations
        $f$ and $g$ over $\{0,1\}^k$ along with their trapdoor information $t_f$ and $t_g$ by running
        algorithm GENT on input $1^k$;
     2. set $\mathtt{Pre} = (f, g)$ and $\mathtt{PreAux} = (R, t_f, t_g)$;
    return $(\mathtt{Pre}, \mathtt{PreAux})$.
Receive random string   $\Sigma = \Sigma_1 \circ \Sigma_2 \circ \Sigma_3$.
Proof Stage
    ALGORITHM $\mathcal{P}_2(x, y, \mathtt{Pre}, \Sigma, \mathtt{PreAux})$ where $(x, y) \in R_L$.
     1. $(\mathtt{comTrap}, \mathtt{decTrap}) = \mathrm{COM}(\Sigma_1, (f, t_f), t_g)$;
     2. $(\mathtt{comWit}, \mathtt{decWit}) = \mathrm{COM}(g, y)$;
     3. $\pi_0 = \mathrm{PROVER}(R; (f, g, \Sigma_1, \mathtt{comTrap}), \Sigma_2, \mathtt{decTrap})$;
     4. $\pi_1 = \mathrm{PROVER}((x, g, \mathtt{comWit}), \Sigma_3, \mathtt{decWit})$;
     5. set $\Pi = (\mathtt{comTrap}, \mathtt{comWit}, \pi_0, \pi_1)$;
    return $\Pi$;
$\mathcal{V}$'s decision
    if the verifier algorithm VERIFIER of each of the two one-round NIZK proofs of
    membership and the verify algorithm VERT of the certified one-way trapdoor
    permutations $f$ and $g$ output true then output true, otherwise output false.
```

Fig. 1. The Double-Round NIZKPK for NP

hiding property does not hold anymore. Consequently, the protocol is neither unbounded zero-knowledge nor even unbounded witness indistinguishable. What is surprising is that this simplified protocol still enjoys the single-theorem zero-knowledge property and this could be (incorrectly) considered sufficient for the transformation of [6].

Non-Malleable NIZKPKs. We also show that the Double-Round NIZKPK of Figure 1 is a Double-Round unbounded NIWIPK. Moreover, following the construction of [6] for the SRS model, we show how to obtain a Double-Round unbounded NMNIZK proof of knowledge starting from a Double-Round unbounded NIWIPK. For the lack of space, details are found in [20].

Acknowledgments

The work described in this paper has been supported in part by the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT and the FP6 program under contract FP6-1596 AEOLUS.

References

1. B. Barak. Constant-Round Coin-Tossing with a Man in the Middle or Realizing the Shared Random String Model. In *FOCS '02*, pp. 345–355, 2002.
2. B. Barak, R. Canetti, J. Nielsen, and R. Pass. Universally Composable Protocols with Relaxed Set-up Assumptions. In *FOCS '04*, pp. 394–403, 2004.

3. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *Eurocrypt '03*, vol. 2045 of *LNCS*, pp. 614–629. Springer-Verlag, 2003.
4. M. Bellare and M. Yung. Certifying Cryptographic Tools: The Case of Trapdoor Permutations. In *Crypto '92*, vol. 740 of *LNCS*, pp. 442–460. Springer Verlag, 1993.
5. M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-Interactive Zero-Knowledge. *SIAM J. on Computing*, 20(6):1084–1118, 1991.
6. A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-Interactive Zero Knowledge. In *Crypto '01*, vol. 2139 of *LNCS*, pp. 566–598. Springer-Verlag, 2001.
7. A. De Santis, G. Di Crescenzo, and G. Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all np relations. In *ICALP 00*, vol. 1853 of *LNCS*, pp. 451–462. Springer Verlag, 2000.
8. A. De Santis and G. Persiano. Zero-Knowledge Proofs of Knowledge Without Interaction. In *FOCS '92*, pp. 427–436, 1992.
9. D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM J. on Computing*, 30(2):391–437, 2000.
10. C. Dwork and M. Naor. Zaps and their Applications. In *FOCS '00*, pp. 283–293, IEEE Computer Society Press, 2000.
11. U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero Knowledge Proofs Under General Assumptions. *SIAM J. on Computing*, 29(1):1–28, 1999.
12. U. Feige and A. Shamir. Witness Indistinguishable and Witness Hiding Protocols. In *STOC '90*, pp. 416–426. ACM, 1990.
13. O. Goldreich and L. Levin. A Hard-Core Predicate for all One-Way Functions. In *STOC '89*, pp. 25–32, 1989.
14. O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, 38(3):691–729, 1991.
15. J. Katz, R. Ostrovsky, and A. Smith. Round Efficiency of Multi-Party Computation with a Dishonest Majority. In *Eurocrypt '03*, vol. 2045 of *LNCS*, pp. 578–595. Springer-Verlag, 2003.
16. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential Aggregate Signatures from Trapdoor Permutations. In *Eurocrypt '04*, vol. 3027 of *LNCS*, pp. 74–90. Springer-Verlag, 2004.
17. P. MacKenzie and K. Yang. On Simulation-Sound Trapdoor Commitments. In *Eurocrypt '04*, vol. 3027 of *LNCS*, pp. 382–400. Springer-Verlag, 2004.
18. S. Micali and L. Reyzin. Min-Round Resettable Zero-Knowledge in the Public-key Model. In *Eurocrypt '01*, vol. 2045 of *LNCS*, pp. 373–393. Springer-Verlag, 2001.
19. R. Ostrovsky and A. Wigderson. One-Way Functions are Essential for Non-Trivial Zero Knowledge. In *ISTCS '93*, pp. 3–17. IEEE Computer Society Press, 1993.
20. G. Persiano and I. Visconti. On Non-Interactive Zero-Knowledge Proofs of Knowledge in the Shared Random String Model. Full version, available at `http://www.dia.unisa.it/professori/visconti/dr.pdf`, 2006.
21. A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. In *FOCS '99*, pp. 543–553, IEEE Computer Society Press, 1993.

Constrained Minimum Enclosing Circle with Center on a Query Line Segment

Sasanka Roy[1], Arindam Karmakar[1], Sandip Das[2], and Subhas C. Nandy[1]

[1]Indian Statistical Institute, Calcutta - 700108, India
[2]Institut de Mathmatiques de Bourgogne, Dijon - 21078, France

Abstract. In this paper, we will study the problem of locating the center of smallest enclosing circle of a set P of n points, where the center is constrained to lie on a query line segment. The preprocessing time and space complexities of our proposed algorithm are $O(n \log n)$ and $O(n)$ respectively; the query time complexity is $O(\log^2 n)$. We will use this method for solving the following problem proposed by Bose and Wang [3] - given r simple polygons with a total of m vertices along with the point set P, compute the smallest enclosing circle of P whose center lies in one of the r polygons. This can be solved in $O(n \log n + m \log^2 n)$ time using our method in a much simpler way than [3]; the time complexity of the problem is also being improved.

1 Introduction

The smallest enclosing circle problem was originally posed in 1857 by Sylvester [13]. Here, a set P of n points are distributed on a 2D plane, and the objective is to identify a point c (not necessarily a member of P) such that the maximum Euclidean distance of the members of P from c is minimum among all other points on the plane. In other words, here the objective is to report the center of the minimum radius circle which can enclose all the points in P. An $O(n^2)$ time algorithm was proposed long ago by Elzinga and Hearn [5]. Later, Shamos and Hoey [12] and Preparata [10] independently proposed algorithms for this problem; each of them runs in $O(n \log n)$ time. Lee [8] proposed the furthest point Voronoi diagram, which can also be used for solving this problem in $O(n \log n)$ time. Finally Megiddo [9] proposed an optimal $O(n)$ time algorithm for solving this problem using prune-and-search technique.

Several constrained versions of the smallest enclosing circle problem are also available in the literature. Megiddo [9] studied the case where the center of the smallest enclosing circle of the point set P is constrained to lie on a given straight line. The time complexity of this algorithm is $O(n)$. Hurtado, et al. [6] provided an $O(n + m)$ time algorithm for finding the smallest enclosing circle of n points such that its center lies inside a convex polygon of size m. Bose et al. [2] considered the generalized version of the problem where the center of the smallest enclosing circle of P is constrained to lie inside a given simple polygon of size m. Their proposed algorithm runs in $O((n + m) \log(n + m) + k)$

time, where k is the number of intersections of the boundary of the polygon with the furthest point Voronoi diagram of P. In the worst case, k may be $O(n^2)$. This result is later improved to $O((n+m)\log m + m\log n)$ [3]. In a further generalization of this problem, r (≥ 1) simple polygons with a total of m vertices are given; one of them can contain the center of the smallest enclosing circle of the point set P [3]. The time complexity of this version is $O((n+m)\log n + (n\sqrt{r}+m)\log m + m\sqrt{r} + r^{\frac{3}{2}}\log r)$.

In this work, we will consider the online query version of the problem proposed in [9]. Given the point set P, we preprocess them such that given any arbitrary query line segment L, the smallest enclosing circle with center on L can be reported efficiently. We first consider that the query object L as a line, and propose two algorithms for this problem. The first one explains the idea of our method. The preprocessing time and space of this method are both $O(n\log n)$, and the query time complexity is $O(\log^2 n)$. The second one uses a complicated idea to create the data structure such that the space complexity is reduced to $O(n)$ keeping the preprocessing and query time complexities unchanged. Next, we show that the same method works when the query object L is a line segment with the same complexity results.

We show that our algorithm can be used for solving the following problem: *Given a set P of n points, and r simple polygons with a total of m vertices. Compute a smallest enclosing circle of P whose center lies inside one of these r polygons.* Our proposed algorithm is simple, and it solves this problem in $O(n\log n + m\log^2 n)$ time, Thus, our algorithm is an improvement of the time complexity of the problem over that of [3].

2 Basic Results

It can be easily observed that, the smallest circle enclosing the vertices of the convex hull of a point set P will also enclose all the points in P. We assume that the vertices of the convex hull of P are ordered in anticlockwise direction. We first describe the role of furthest point Voronoi diagram in computing the smallest enclosing circle of P, and then use it to solve our online query problem.

Let $\mathcal{V}(P)$ denote the furthest point Voronoi diagram of P. It partitions the plane into n unbounded convex regions, namely $R(p_1), R(p_2), \ldots, R(p_n)$, such that for any point $p \in R(p_j)$, $\delta(p, p_j) > \delta(p, p_k)$ for all $k = 1, 2, \ldots, n$, and $k \neq j$. Here $\delta(.,.)$ denotes the Euclidean distance between a pair of points. Computing the convex hull of P needs $O(n\log n)$ time and $O(n)$ space, and then computing $\mathcal{V}(P)$ needs $O(n)$ time and space [1]. Given a query point q, this data structure can report the region $R(p_i)$ containing q in $O(\log n)$ time.

Lemma 1. *[8] In the unconstrained situation, the smallest enclosing circle of P always passes through at least two points of P.*

Lemma 1 says that, the center c of the unconstrained smallest enclosing circle of P always lies on an edge e of $\mathcal{V}(P)$. In [8], an algorithm is proposed for computing c; this runs in $O(n\log n)$ time. We will use this information in designing the constrained smallest enclosing circle of P whose center c' lies on a given query line segment L. We first develop the algorithm for the case where the query object L is a line. Next, we show that a minor modification of that algorithm works when L is a line segment.

In our discussion, we will use C and C' to denote the unconstrained and constrained smallest enclosing circles respectively; c and c' denote the center of C and C' respectively (see Fig. 1(a)). If ρ and ρ' denote the radius of C and C' respectively, then $\rho \leq \rho'$. As opposed to the fact that C must pass through at least two points of P (see Lemma 1), C' may pass through only one point in P. The following observation lists distinct cases to be considered for computing c'.

Observation 1. *For a convex polygon P,*

(a) if C' passes through only a single point $p \in P$ then p is the furthest point from the query line L, and the perpendicular projection of p on the line L (denoted by p') lies inside $R(p)$, and vice-versa. Here, $p' = c'$ is the center of the circle C'.

(b) if C' passes through exactly two points $p_1, p_2 \in P$ then its center c' is the intersection point of L with an edge e of $\mathcal{V}(P)$, where e is on the perpendicular bisector of p_1 and p_2.

(c) if C' passes through more than two points of P, then c' must be a vertex of $\mathcal{V}(P)$, and this vertex lies on the line L.

In order to test whether C' passes through a single point in P, we perform the following steps:

1. Identify the point $p \in P$ which is farthest from L.
2. Compute the perpendicular projection of p on L. Let this point be $\hat{c}$.
3. Identify the point $q \in P$ whose Voronoi region $R(q)$ contains the point $\hat{c}$.
4. If $q = p$ then C' passes through p, and hence $c' = \hat{c}$.

The query time in step 1 is $O(\log n)$ time [4] with a preprocessing, which needs $O(n \log n)$ time and $O(n)$ space. Step 2 can be done in constant time. For step 3, the point location can be performed in $O(\log n)$ time using an $O(n)$ size data structure for $\mathcal{V}(P)$, which can be constructed in $O(n \log n)$ time [11]. Thus, we have the following lemma.

Lemma 2. *If C' passes through only a single point P, then its center c' can be computed in $O(\log n)$ time using a preprocessed data structure of size $O(n)$, which can be constructed in $O(n \log n)$ time.*

If the above test fails, then C' passes through two or more points of P as mentioned in Observation 1. Here, the center c' of the circle C' will be an intersection point of L and an edge of $\mathcal{V}(P)$. In the degenerate case, this point may be a vertex of $\mathcal{V}(P)$, indicating that C' passes through more than two points of P.

Let L intersect the edges $e_1, e_2, \ldots, e_m$ of $\mathcal{V}(P)$ in order, and a_i denote the intersection of L with e_i. The above discussions say that c' will coincide with a member in $A = \{a_1, a_2, \ldots, a_m\}$. Let $\rho(a_i)$ denote the radius of the smallest enclosing circle of P with center at a_i.

Lemma 3. *The sequence* $\{\rho(a_1), \rho(a_2), \ldots, \rho(a_m)\}$ *is unimodal.*

Proof. Let $\rho(a_k) = \min_{i=1}^{m} \rho(a_i)$. We show that $\rho(a_j) < \rho(a_{j+1})$ for $j = k, k+1, \ldots, m-1$. Similarly, it can be shown that $\rho(a_{j'}) < \rho(a_{j'-1})$ for $j' = k, k-1, \ldots, 2, 1$ (see Fig. 1(a)).

Draw a line ℓ perpendicular to L at the point a_k. If e_k is the Voronoi edge corresponding to points $p, p' \in P$, then p and p' lies on different sides of ℓ, and $\delta(p, a_k) = \delta(p', a_k)$. Let p (resp. p') be to the right (resp. left) of ℓ. Now, for any point α in the interval $[a_k, a_{k+1}]$ (on the line L), $\alpha \in R(p')$ and $\delta(\alpha, p') > \delta(a_k, p')$. Thus, we have $\rho(a_{k+1}) > \rho(a_k)$. Next we prove that $\rho(a_{k+1}) < \rho(a_{k+2})$.

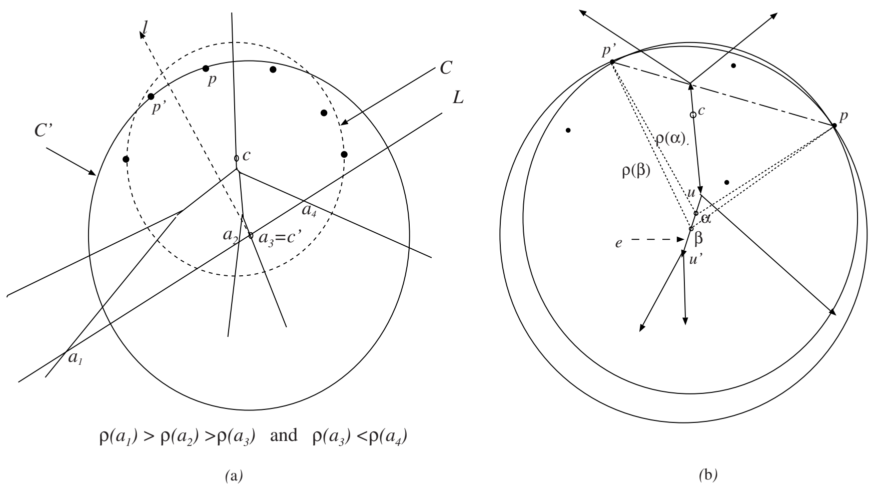


Fig. 1. (a) Proof of Lemma 3, and (b) Proof of Lemma 4

Let e_{k+1} be the Voronoi edge corresponding to p' and p''. Choose a point $\gamma \in [a_{k+1}, a_{k+2}]$ on the line L. Using the same argument, it can be shown that $\rho(a_{k+1}) = \delta(p', a_{k+1}) = \delta(p'', a_{k+1}) < \delta(p'', \gamma)$. Thus, $\rho(\gamma) > \rho(a_{k+1})$ since the circle enclosing P with center at γ has to enclose p''. Proceeding similarly, we can prove the other inequality results. $\square$

It is already mentioned that each cell of $\mathcal{V}(P)$ is an unbounded convex region and the center c of the unconstrained smallest enclosing circle C lies on an edge of $\mathcal{V}(P)$. Thus, $\mathcal{V}(P)$ may be viewed as a directed tree $\mathcal{T}$ with c as root node, and all the Voronoi vertices are the internal nodes of $\mathcal{T}$ (see Fig. 1(a)). The leaf nodes are hypothetical in the sense that these are at the open ends of the half-line edges. In order to clearly define the leaf-nodes of $\mathcal{T}$, we consider an axes-parallel square which contains all the vertices of $\mathcal{V}(P)$, and observe its intersections with all the unbounded edges of $\mathcal{V}(P)$. Let these be $E = \{\eta_1, \eta_2, \ldots, \eta_n\}$ in anticlockwise order along the boundary of the square. These will serve the role of leaf nodes in $\mathcal{T}$. We will use $\pi(v)$ to denote the directed path from c to v in $\mathcal{T}$. We will also use $depth(v)$ to denote the number of nodes on the path $\pi(v)$.

Lemma 4. *If T_u denotes the subtree rooted at an internal node $u \in \mathcal{T}$, then $\rho(v) > \rho(u)$ for each vertex $v \in T_u$, $v \neq u$.*

Proof. Let u' be a successor of u in $\mathcal{T}$. We need to prove $\rho(u) < \rho(u')$. Consider the directed edge $e = (u \rightarrow u')$. The regions adjacent to e are $R(p)$ and $R(p')$ respectively, where $p, p' \in P$. Thus, the edge e is the perpendicular bisector of the line segment $[p, p']$. At any point $\alpha \in e$, $\rho(\alpha) = \delta(\alpha, p) = \delta(\alpha, p')$. Moreover, for a pair of points $\alpha, \beta \in e$, if $\delta(\alpha, u) < \delta(\beta, u)$, then $\rho(\alpha) < \rho(\beta)$ because of the fact that both the triangles $\Delta pp'\alpha$ and $\Delta pp'\beta$ are isosceles, having the same base, and the other vertex is moving away from the base along the perpendicular bisector of the base. Thus, $\Delta pp'\alpha$ is inside $\Delta pp'\beta$ (see Fig. 1(b)). Thus, $\rho(u') > \rho(u)$. Applying the same technique recursively, the lemma follows. □

Lemma 4 says that as we go far from c along a path in $\mathcal{T}$, the ρ value of the nodes along that path increases monotonically. Thus, if a Voronoi region R contains c as its vertex, then $\rho(c)$ is minimum among the ρ values of all other vertices in R. If R does not have c as one of its vertices, and $\rho(w) = min_{v \in R}\ \rho(v)$, then for every vertex $v \in R$, the path from c to v in $\mathcal{T}$ passes through node w. In addition, the following result is an implication of Lemma 4.

Corollary 4.1. *If L intersects a path $\pi(\eta_i)$ more than once, then the intersection point having minimum depth is the candidate for being c'.*

3 Constrained Smallest Enclosing Circle Problem with Center on a Query Line

In our actual problem, the vertices of a convex polygon P are given in anticlockwise order. We propose two methods for this problem. The first one is simple but it takes $O(n \log n)$ space. It gives a clear idea about our method. In the next one, we adopted a complicated pointer structure to reduce the space complexity to $O(n)$. For both the methods, the preprocessing time complexity is $O(n \log n)$, and the query time complexity is $O(\log^2 n)$.

Suppose the furthest point Voronoi diagram $\mathcal{V}(P)$ of the polygon P is already computed, and is stored in the form of a directed tree $\mathcal{T}$ with root at c. Each node v is attached with its $\rho(v)$ value, which can be easily computed by observing the cells of $\mathcal{V}(P)$ in which v belongs. In addition, each node is attached with a *parent_pointer* which points to its predecessor in $\mathcal{T}$. The set E of leaf nodes of $\mathcal{T}$ are also stored in an array, and each element in E points to its corresponding element in $\mathcal{T}$.

3.1 Method-1

Preprocessing

In this method, we attach a few pointers with each node of $\mathcal{T}$ by performing a depth first search on the tree $\mathcal{T}$ in the preprocessing phase. Let v be a node with *depth*$(v) = m$. We attach a secondary structure B_v with node v which is an array of size $\lceil \log m \rceil$. These will contain the address of the nodes at depth $\lceil \frac{m}{2} \rceil$, $\lceil \frac{3m}{4} \rceil$, $\lceil \frac{7m}{8} \rceil$, ... respectively along the path $\pi(v)$ in the mentioned order (see Fig. 2(a)). The computation of these pointers are described below.

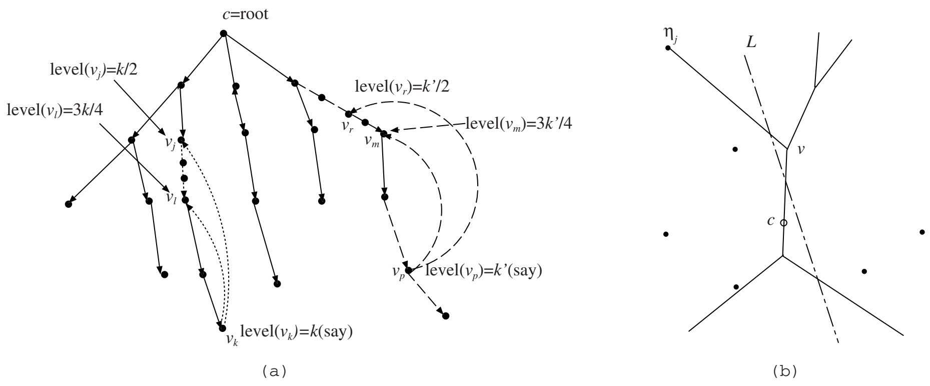


Fig. 2. (a) Intuitive idea of the secondary structures, and (b) Illustration of Lemma 6

We implement the stack required for the depth first search using an array. During the depth first search when the path follows a forward edge, we push the address of the corresponding node in the stack. While backtracking from a node v, we create the secondary structure B_v, and then pop v from the stack.

Lemma 5. *The Preprocessing phase can be completed in $O(n \log n)$ time and using $O(n \log n)$ space.*

Proof. The furthest point Voronoi diagram $\mathcal{V}(P)$ can be computed in $O(n \log n)$ time [11]. The computation of c needs $O(n)$ time [9]. Assigning the direction of the edges in $\mathcal{T}$ needs another $O(n)$ time. Finally, the depth first search needs $O(n)$ time. While processing a node v during the backtrack, the creation of the array B_v of pointers needs $O(\log n)$ time in the worst case because $depth(v) \leq n$. The space complexity also follows from the same argument. $\square$

Query answering

Given a query line L, we identify two paths $\pi(\eta_k), \pi(\eta_{k'})$, where $\eta_k, \eta_{k'} \in E$ such that the leaf nodes of all the paths $\Pi_1 = \{\pi(\eta_{k+1}), \pi(\eta_{k+2}), \ldots \pi(\eta_{k'-1})\}$ lie in the opposite side of c with respect to L, and the leaf nodes of all the paths in $\Pi_2 = \{\pi(\eta_{k'}), \pi(\eta_{k'+1}), \ldots \pi(\eta_{k-1}), \pi(\eta_k)\}$ lie in the same side of c with respect to L. Note that, all the paths in Π_1 are intersected by the line L. The paths $\pi(\eta_k)$ and $\pi(\eta_{k'})$ can be identified in $O(\log n)$ time using the array E.

Lemma 6. *The center c' of the circle C' must be an intersection point of L with one of the paths in Π_1.*

Proof. It is already mentioned in Observation 1 (b) & (c) that the point c' lies on a path of $\Pi_1 \cup \Pi_2$. We need to prove that if c' is observed on a path in Π_2, then it must also lie on some path of Π_1 also.

Consider an η_j such that the path $\pi(\eta_j) \in \Pi_2$ and c' lies on $\pi(\eta_j)$. By the definition of Π_2, the points c and η_j are in the same side of L. Since L intersects the path $\pi(\eta_j)$, there exists a vertex (say $v \in \mathcal{T}$) which lies in the opposite side of η_j with respect to L. Now, consider the path $\pi(v)$, where c and v lies in different sides of L (see Fig. 2(b)). This path has also been intersected by L. As Π_1 is non-empty and the Voronoi regions are convex, there exists a path in Π_1 passing through v. This proves the lemma. □

Lemma 7. *If the line L intersects a path $\pi(\eta_j)$ multiple times, and the center c' lies on $\pi(\eta_j)$, then c' will be the intersection point which is closest to c along the path $\pi(\eta_j)$.*

Proof. Follows from Corollary 4.1. □

Searching for an intersection of L with a path

The following lemma says that the secondary structures attached to the nodes in a path helps us in searching for an intersection point of that path with line L.

Lemma 8. *The worst case time complexity of searching for an intersection point of L with the path $\pi(\eta_j)$ is $O(\log n)$.*

Proof. Let $depth(\eta_j) = m$. We refer to the nodes on the path $\pi(\eta_j)$ as $v_1(= c), v_2, \ldots, v_m(= \eta_j)$. We first consider the first link in B_{v_m}. This points to the $v_{\lceil \frac{m}{2} \rceil}$-th node on that path. If c and $v_{\lceil \frac{m}{2} \rceil}$ are in the same side of L, then L has at least one intersection in the sub-path from $v_{\lceil \frac{m}{2} \rceil}$ and v_m; otherwise it has intersected $\pi(v_{\lceil \frac{m}{2} \rceil})$. In the former case, we need to observe the next link of v_m, and in the latter case, we need to observe the first link in $B_{v_{\lceil \frac{m}{2} \rceil}}$. Proceeding this way, we can easily identify an edge on the path $\pi(\eta_j)$ of $\mathcal{T}$ which has been intersected with L. This needs $O(\lceil \log m \rceil)$ time. □

Searching for c' along L

Let $A = \{a_1, a_2, \ldots, a_\mu\}$ be a sequence of intersection points of L with the edges of $\mathcal{T}$, where a_j lies on path $\pi(\eta_j)$. Let $c' = a_i$. As the sequence of ρ values of the members in A is unimodal (see Lemma 3), both the sub-sequences $\{\rho(a_i), \rho(a_{i-1}), \ldots \rho(a_1)\}$ and $\{\rho(a_i), \rho(a_{i+1}), \ldots \rho(a_\mu)\}$ are monotonically increasing. So, we can identify c' by performing a binary search among the members in A. While considering a path $\pi(\eta_j)$, we compute a_j as mentioned in the earlier subsection. Next, we compute $\rho(q)$ and $\rho(q')$ for a pair of points q and q' at a distance ϵ $(\leq \min(\delta(a_j, a_{j-1}), \delta(a_j, a_{j+1})))$ from a_j on line L. This helps us to decide whether $c' = a_j$ or c' is towards left or right from a_j along L.

Note that L may intersect a path, say $\pi(\eta_j)$ many times. If $c' \in \pi(\eta_j)$, then by Lemma 7, it is the point of intersection of L and the path $\pi(\eta_j)$ which is closest to c along $\pi(\eta_j)$. Suppose we have a situation where $a_j \neq c'$ but $c' \in \pi(\eta_j)$. The following lemma says that c' will also lie on some other path, say $\pi(\eta_k)$. Finally, we will show that our algorithm will also explore the path $\pi(\eta_k)$, and will identify c'.

Lemma 9. *If L intersects the path $\pi(\eta_j)$ more than once, say at q_1 and q_2, and $depth(q_1) < depth(q_2)$, then (i) there exists another intersection point q (may be q_1 itself) which lies on some other path, say $\pi(\eta_\ell) \in \Pi_1$, on which q_2 does not lie, and (ii) $\rho(q) \leq \rho(q_1) < \rho(q_2)$ (equality holds if $q = q_1$).*

Proof. Similar to the proof of Lemma 6. □

As the paths in Π_1 are non-crossing, the entire path of $\pi(\eta_\ell)$ starting from q lies in one side of the path $\pi(\eta_j)$. This leads to the fact that the members of E corresponding to the paths in Π_1 are in the same order as the order of the intersection points of L with the corresponding paths.

We perform binary search among the members in E. For each choice η_j, we compute the intersection point a_j on $\pi(\eta_j)$, and then check whether $a_j = c'$ or we need to move towards left (resp. right) on L by computing ρ values of two points on L in the ϵ neighborhood of a_j. Thus, we have the following theorem stating the complexity results of Method-1.

Theorem 1. *Method-1 correctly computes c' with $O(n \log n)$ preprocessing time and space complexities, and with worst case query time complexity $O(\log^2 n)$.*

Proof. The correctness of our proposed algorithm follows from Lemma 9. The preprocessing time and space complexity results follow from Lemma 5. The query time complexity follows from Lemma 8 and the fact that we may need to compute $O(\log n)$ elements of A to find c'. □

3.2 Method-2

We now describe a different method of creating the secondary structure for each node in $\mathcal{T}$. This will reduce the space complexity of the problem to $O(n)$ keeping the preprocessing and query time complexities unchanged.

Revised secondary structure
Instead of keeping $O(\log n)$ pointers as the secondary structure of each node in $\mathcal{T}$, we store only two link fields, namely *ptr_1* and *ptr_2* with each node in $\mathcal{T}$. The ptr_1 pointer attached to a node $v \in \pi(\eta_j)$ indicates that while searching for an intersection of L with the path $\pi(\eta_j)$, if node v is reached, then such an intersection point must be observed in the path segment between the nodes v and $v.ptr_1$. The ptr_2 pointer of node v points to the middlemost node in the path segment between v and $v.ptr_1$. In order to set these two pointers of the nodes in $\mathcal{T}$, we have to create a temporary data structure as mentioned below.

Let the maximum depth of a leaf node in $\mathcal{T}$ be m^*. We first create a temporary array A of size 2^α, where $2^{\alpha-1} < m^* \leq 2^\alpha$. Each entry of the array A consists of two fields, which are also named as *ptr_1* and *ptr_2* respectively.

Consider a path of length 2^α whose nodes are $c = v_1, v_2, v_3, \ldots, v_{2^\alpha} = \eta$. As mentioned earlier, the search in a path of $\mathcal{T}$ starts from its leaf node, and L may intersect any edge on that path. A portion of the path is indicated by the corresponding interval of node-indices. Thus, initially we have the interval $[2^0, 2^\alpha]$. We use a stack whose each element is a tuple of the form (I, i), where I is

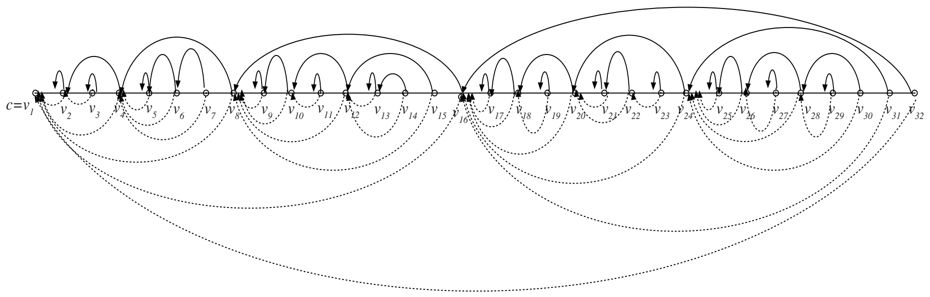


Fig. 3. Secondary structures of the nodes in $\mathcal{T}$

an interval of node-indices, and i is an integer ($0 \leq i \leq \log n$). Initially, we push the tuple $([2^0, 2^\alpha], 0)$ onto the stack. Each time we pop an element $([a, b], i)$ from stack. Set *ptr_1* and *ptr_2* of $A[b-i]$ to a and $\frac{a+b}{2}$ respectively. If $a+1 = b-i$, then *ptr_2* of $A[b-i]$ is set to $b-i$ itself (see Fig. 3). If $a+1 \neq b-i$, the interval $I = [a, b]$ gets split into two sub-intervals of node indices, namely $I_1 = [a, \frac{a+b}{2}]$ and $I_2 = [\frac{a+b}{2}, b]$. We push both the tuples $(I_1, 0)$ and $(I_2, i+1)$ in the stack. The process continues until the stack becomes empty. The creation of the array A is clearly illustrated in Fig. 3. Here ptr_1 and ptr_2 pointers of that node are represented using dashed and solid edges respectively.

After the creation of the array A, we will set ptr_1 and ptr_2 of each node v by performing a depth first search as in Method-1. Here, while popping a node v from the i-th position of the stack, we will set $v.ptr_1$ and $v.ptr_2$ by $A[i].ptr_1$ and $A[i].ptr_2$ respectively.

Lemma 10. *If a path $\pi(\eta)$ from c to a leaf node η in $\mathcal{T}$ is of length 2^β ($\beta \leq \alpha$), then the aforesaid link setting can report an intersection point of L with $\pi(\eta)$ in $O(\beta)$ time.*

Proof. We will use only *ptr_2* to prove this lemma. The role of pointer *ptr_1* will be clear in the next subsection.

Let $depth(\eta) = 2^\alpha$. As in Method-1, α pointers are available to η, which are stored in the ptr_2 fields of α nodes from η towards the root. So, here we can use these pointers moving upwards from η. In addition, while visiting these nodes, it has checked whether L has intersected the edges attached to these α nodes. The pointers which were present in the secondary structures of these $\alpha - 1$ nodes in Method-1, are not necessary in this method.

Suppose, after processing β ($\leq \alpha$) nodes starting from η, we could identify a node having depth $2^{\alpha-1} + 2^{\alpha-2} + \ldots + 2^{\alpha-\beta}$ such that L has intersected an edge in the sub-path from $v_{(2^{\alpha-1}+2^{\alpha-2}+\ldots+2^{\alpha-\beta-1})}$ to $v_{(2^{\alpha-1}+2^{\alpha-2}+\ldots+2^{\alpha-\beta})}$. The search interval is further pruned using the $\alpha - \beta$ pointers attached to $v_{(2^{\alpha-1}+2^{\alpha-2}+\ldots+2^{\alpha-\beta})}$. The process continues until the edge of $\pi(\eta)$ is identified which has been intersected by L. The result follows from the fact that, the total number of link traversals in this process is at most α.

Next, consider the case where $depth(\eta) = 2^\beta$, $\beta < \alpha$. Here, observe that the similar link structure is maintained among the nodes in $v_1, v_2, \ldots, v_{2^\beta}$ (see Fig. 3). Thus, the result follows. □

Searching in a path
Consider a path $\pi(\eta)$, where η is a leaf node and $depth(\eta) = m$, where $2^{\beta-1} < m \leq 2^\beta$, and $\beta \leq \alpha$. Without loss of generality, assume that the nodes are named as $c = v_1, v_2, \ldots, v_m = \eta$, where $m = 2^{\beta-1} + 2^{\beta-2} + \ldots + 2^{\beta-j} + m'$, and $m' < 2^{\beta-j-1}$. Suppose the query line L intersects the edge $(v_\ell, v_{\ell+1}) \in \pi(\eta)$. Here we need to consider two cases depending on whether (1) L intersects the path $\pi(\eta)$ between v_0 and $v_{2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-j}}$, or (2) L intersects the path $\pi(\eta)$ below $v_{2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-j}}$.

Case 1: We use the following notations to describe our search algorithm. Let $\theta = 2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-k}$ and $\theta' = \theta - 2^{\beta-k} = 2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-k-1}$. Our search starts from v_m, and it consists of two major tasks: (i) identify θ such that v_ℓ lies in the interval of node-indices $[v_\theta, v_{\theta'}]$, and (ii) search for the intersection in the interval of node indices $[v_\theta, v_{\theta'}]$.
The task (i) is performed using *ptr*_1. Task (ii) is done using a binary search using *ptr*_2. Let us now observe Fig. 3 to understand the search technique. Consider a typical instance where the path length is $m = 29$, and assume that L has intersected the edge $(v_\ell, v_{\ell+1})$, where $\ell = 2$. Thus, here $\theta = 16$ and $\theta' = 1$. The query involves the following link traversals (i) $v_{29} \rightarrow v_{28} \rightarrow v_{24} \rightarrow v_{16}$ using pointer *ptr*_1, and then (ii) we apply binary search (as mentioned in Lemma 10) in the interval $[v_{16}, v_1]$ using *ptr*_2 to reach the node v_3. The intersection point of L with the edge (v_1, v_2) will then be identified.

Case 2: If L intersects an edge below $v_{2^{\beta-1}+2^{\beta-2}+\ldots+2^{\beta-j}}$, then it can also be detected in $O(\log n)$ time by expressing m' as that of m.

Theorem 2. *The preprocessing time and space complexities of Method-2 are $O(n \log n)$ and $O(n)$ respectively, and the worst case query time complexity is $O(\log^2 n)$.*

Proof. In addition to the preprocessing steps of Method-1, we had to create the array A containing the node indices on a path. This step needs $O(2^\alpha)$ time, if the length of the longest path m^* lies in $2^{\alpha-1} < m^* \leq 2^\alpha$. The secondary structure of each node is of size 2, and it needs an additional $O(1)$ time for each node in $\mathcal{T}$ while executing the depth first search. Thus both the preprocessing time and space complexity results follow.

In the worst case query time complexity analysis, we will study the search time on a path only. The location of c' among the possible paths remain same as in Method-1. The searching in a path in this method consists of two parts: (i) the number hops needed to reach from v_m to v_θ, and (ii) the time needed for the binary search to reach from v_θ to v_ℓ using *ptr*_2, Both the steps need $O(\alpha)$ hops. Since, we may need to inspect $O(\log n)$ paths in the worst case (see Lemma 8), the result follows. □

4 Constrained Smallest Enclosing Circle Problem with Center on a Query Line Segment

We now consider that the query object L is a line segment. Let $\hat{L}$ be the line containing the line segment L. We apply Method-2 to identify the center $\hat{c} \in \hat{L}$ of the constrained smallest enclosing circle. If $\hat{c}$ is observed to be inside L, then report $c' = \hat{c}$. Otherwise, by Lemma 3, the center of the desired constrained smallest enclosing circle is one of the endpoints of L which is closest to $\hat{c}$ (with respect to the Euclidean distance). Finally, the radius of the desired smallest enclosing circle is $\delta(p, c')$, where $p \in P$ is the point whose corresponding Voronoi cell contains c'.

5 Constrained Smallest Enclosing Circle Problem with Center in a Given Set of Polygons

Here the query objects are r simple polygons with a total of m edges. We first compute the furthest point Voronoi diagram $\mathcal{V}(P)$, to identify the center c of the unconstrained smallest enclosing circle. If it is inside one of these polygons, we report the answer. Otherwise, the center will be on the boundary of one of these polygons. For each edge (line segment), We compute the center of the constrained smallest enclosing circle with center on that edge, and report the radius of the smallest one. Thus, the overall time complexity becomes $O(n \log n + m \log^2 n)$, where $|P| = n$. In [2], it is mentioned that there may exist more than one such circle attaining the smallest radius. Our algorithm can report all these circles with the same time complexity.

References

1. A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor, *A linear-time algorithm for computing the Voronoi diagram of a convex polygon*, Discrete Computational Geometry, vol. 4, pp. 591-604, 1989.
2. P. Bose and G. Toussaint, *Computing the constrained Euclidean, geodesic and link center of a simple polygon with applications*, Studies of facility location analysis, vol. 15, pp. 37-66, 2000.
3. P. Bose and Q. Wang, *Facility location constrained to a polygonal domain*, Theoretical Informatics, 5th Latin American Symposium, LNCS 2286, pp. 153-164, 2002.
4. O. Daescu, N. Mi, C. Shin and A. Wolff, *Furthest-point queries with geometric and combinatorial constraints*, Computational Geometry: Theory and Applications, to appear in 2006.
5. J. Elzinga and D. W. Hearn, *Geometrical solutions to some minimax location problems*, Transpotation Science, vol. 6, pp. 379-394, 1972.
6. F. Hurtado, V. Sacristan and G. Toussaint, *Facility location problems with constraints*, Studies in Locational Analysis, vol. 15, pp. 17-35, 2000.
7. D.T. Lee and Y.T. Ching. *The power of geometric duality revised*, Information Processing Letters, vol. 21, pp. 117-122, 1985.

8. D.T. Lee, *Furthest neighbour Voronoi diagrams and applications*, Report 80-11-FC-04, Dept. Elect. Engrg. Comput. Sci., Northwestern Univ., Evanston, IL, 1980.
9. N. Megiddo, *Linear-time algorithms for linear programming in R^3 and related problems*, SIAM J. Comput., vol. 12, pp. 759-776, 1983.
10. F. P. Preparata, *Minimum spanning circle*, Technical report, Univ. Illinois, Urbana, IL, in Steps into Computational Geometry, 1977.
11. F. P. Preparata and M. I. Shamos,*Computational Geometry: An Introduction, Second edition,* Springer Verlag, 1990.
12. M.I. Shamos and D. Hoey, *Closest-point problem*, Proc. 16th Annual IEEE Sympos. Found. Comput. Sci., pages 151-162, 1975.
13. J. J. Sylvester, *A question in the geometry of situation*, Quarterly Journal of Mathematices, pp. 1-79, 1857.

Hierarchical Unambiguity

Holger Spakowski[1,⋆] and Rahul Tripathi[2]

[1] Institut für Informatik, Heinrich-Heine-Universität Düsseldorf, 40225 Düsseldorf, Germany
spakowsk@cs.uni-duesseldorf.de
[2] Department of Computer Science and Engineering, University of South Florida,
Tampa, FL 33620, USA
tripathi@cse.usf.edu

Abstract. We develop techniques to investigate relativized hierarchical unambiguous computation. We apply our techniques to push forward some known constructs involving relativized unambiguity based complexity classes (UP and Promise-UP) to new constructs involving arbitrary levels of the relativized unambiguous polynomial hierarchy (UPH). Our techniques are developed on constraints imposed by hierarchical assembly of *unambiguous* nondeterministic polynomial-time Turing machines, and so our techniques differ substantially, in applicability and in nature, from standard techniques (such as the switching lemma [Hås87]), which are known to play roles in carrying out similar generalizations.

Aside from achieving these generalizations, we resolve a question posed by Cai, Hemachandra, and Vyskoč [CHV93] on an issue related to nonadaptive Turing access to UP and adaptive smart Turing access to Promise-UP.

1 Introduction

Baker, Gill, and Solovay in their seminal paper [BGS75] introduced the concept of relativization in complexity theory, and showed that the primitive levels of the polynomial hierarchy, i.e. P and NP, separate in some relativized world. Baker and Selman [BS79] made progress in extending this relativized separation—$\mathrm{P} \neq \mathrm{NP}$ in some relativized world—to the next levels of the polynomial hierarchy: They proved that there is a relativized world where $\Sigma_2^p \neq \Pi_2^p$, and so $\Sigma_2^p \neq \Sigma_3^p$ relative to the same world. However, Baker and Selman [BS79] observed that their proof techniques do not apply in achieving relativized separations at higher levels of the polynomial hierarchy because of certain constraints in their counting argument. Thus, it required the development of entirely different proof techniques for separating all the levels of the relativized polynomial hierarchy. The landmark paper by Furst, Saxe, and Sipser [FSS84] established the close connection between the relativization of the polynomial hierarchy and lower bounds for small depth circuits computing certain functions. Techniques for proving such lower bounds were developed in a series of papers [FSS84, Sip83, Yao85, Hås87], which were motivated by questions about the relativized structure of the polynomial hierarchy. Yao [Yao85] finally succeeded in separating the levels of the relativized polynomial

⋆ Supported in part by the DFG under grants RO 1202/9-1 and RO 1202/9-3.

hierarchy by applying these new techniques. Håstad [Hås87] gave the most refined presentation of these techniques via the *switching lemma*. Even to date, Håstad's switching lemma [Hås87] is used as an indispensable tool to separate relativized hierarchies, composed of classes stacked one on top of another. (See, for instance, [Hås87, Ko89, BU98, ST] where the switching lemma is used as a strong tool in proving the feasibility of oracle constructions.) A major contribution of our paper lies in demonstrating that certain known oracle constructions involving the primitive levels of the unambiguous polynomial hierarchy (UPH) and the promise unambiguous polynomial hierarchy ($\mathcal{UPH}$), i.e. UP and $\mathrm{P}_s^{\mathrm{Promise\text{-}UP}}$, respectively, can be extended to oracle constructions that involve arbitrary higher levels of UPH, purely by counting arguments alone. In fact, it seems implausible to achieve these extensions by well-known techniques from circuit complexity (e.g., the switching lemma [Hås87] and the polynomial method surveyed in [Bei93, Reg97]).

The class UP is the unambiguous version of NP. UP has proved to be useful in studying worst-case one-to-one one-way functions [Ko85, GS88] and some closure properties of #P [OH93]. Lange and Rossmanith [LR94] generalized the notion of unambiguity to higher levels of the polynomial hierarchy. They introduced the following unambiguity based hierarchies: AUPH, UPH, and $\mathcal{UPH}$. It is known that AUPH $\subseteq$ UPH $\subseteq$ $\mathcal{UPH}$ $\subseteq$ UAP [LR94, CGRS04], where UAP (unambiguous alternating polynomial-time) is the analog of UP for alternating polynomial-time Turing machines. These hierarchies received renewed interests in some recent papers (see, for instance, [ACRW04, CGRS04, ST, GT05]). Spakowski and Tripathi [ST], developing on circuit complexity-theoretic proof techniques of Sheu and Long [SL96], and of Ko [Ko89], obtained results on the relativized structure of these hierarchies. Spakowski and Tripathi [ST] proved that there is a relativized world where these hierarchies are infinite. They also proved that for each $k \geq 2$, there is a relativized world where these hierarchies collapse so that they have exactly k distinct levels and their k'th levels collapse to PSPACE. The present paper supplements this investigation with a focus on the structure of the unambiguous polynomial hierarchy.

1.1 Results

We prove a combinatorial lemma (Lemma 1) and demonstrate its usefulness in generalizing known relativization results involving classes such as UP and Promise-UP to new relativization results that involve arbitrary levels of the unambiguous polynomial hierarchy (UPH).

In Subsection 4.1, we use Lemma 1 to show that certain inclusion relationships between bounded ambiguity classes ($\mathrm{UP}_{O(1)}$ and FewP) and the levels of the unambiguous polynomial hierarchy (UPH) do not relativize. Theorem 2 of this subsection subsumes an oracle result of Beigel [Bei89] for any constant $k \geq 1$ and Theorem 4 generalizes a result of Cai, Hemachandra, and Vyskoč [CHV93] from the case of $k = 2$ to the case of any arbitrary $k \geq 2$; the parameter k is a part of these theorems.

Subsection 4.2 studies the issue of simulating nonadaptive access to $\mathrm{U}\Sigma_h^p$, the h'th level of the unambiguous polynomial hierarchy, by adaptive access to $\mathrm{U}\Sigma_h^p$. Theorem 6 of this subsection generalizes a result of Cai, Hemachandra, and Vyskoč [CHV92] from the case of $h = 1$ to the case of any arbitrary $h \geq 1$; the parameter h is a part of the theorem. Lemma 1 is used as a key tool in proving Theorem 6.

We improve upon Theorem 6 of Subsection 4.2 in Subsection 4.3. There are compelling reasons for the transition from Subsection 4.2 to Subsection 4.3, which we elaborate in Subsection 4.3. Theorem 8 in that subsection not only resolves a question posed by Cai, Hemachandra, and Vyskoč [CHV93], but also generalizes one of their results. In particular, Theorem 8 holds for any total, polynomial-time computable and polynomially bounded function $k(\cdot)$ and arbitrary $h \geq 1$, while a similar result of Cai, Hemachandra, and Vyskoč [CHV93] holds only for any arbitrary *constant* k and $h = 1$; the parameters k and h are parts of Theorem 8. Lemma 1 is one of the ingredients in the proof of this theorem.

Subsection 4.4 investigates the complimentary issue of simulating adaptive access to $\mathrm{U}\Sigma_h^p$ by nonadaptive access to $\mathrm{U}\Sigma_h^p$. Theorem 9 of this subsection generalizes a result of Cai, Hemachandra, and Vyskoč [CHV93] from the case of $h = 1$ to the case of any arbitrary constant $h \geq 1$. Again, Lemma 1 is useful in making this generalization possible.

In Subsection 4.5, we study the notion of one-sided helping introduced by Ko [Ko87]. Theorem 10 of this subsection generalizes and improves one of the results of Cai, Hemachandra, and Vyskoč [CHV93].

Finally, in Section 5 we consider the possibility of imposing more stringent restriction in the statement of Lemma 1. The investigation in this subsection leads to a generic collapse of $\mathrm{U}\Sigma_k^p$ to P, for each $k \geq 1$, under the assumption P = NP. This generalizes a result of Blum and Impagliazzo [BI87] from the case of $k = 1$ to the case of any arbitrary $k \geq 1$.

Due to the space limit, all proofs are omitted. They will appear in the full version of the paper.

2 Preliminaries

2.1 Notations

Let $\mathbb{N}^+$ denote the set of positive integers. Σ denotes the alphabet $\{0, 1\}$. For every oracle NPTM N, oracle A, and string $x \in \Sigma^*$, we use the shorthand $N^A(x)$ for "the computation tree of N with oracle A on input x." We fix a standard, polynomial-time computable and invertible, one-to-one, multiarity pairing function $\langle ., \ldots, . \rangle$ throughout the paper. Let $\circ$ denote the composition operator on functions. For any polynomial $p(.)$ and integer $i \geq 1$, let $(p\circ)^i(\cdot)$ denote $p \circ p \circ \cdots \circ p(\cdot)$, i.e. the polynomial obtained by i compositions of p.

For any complexity class $\mathcal{C}$ and for any natural notion of polynomial-time reducibility r (e.g., $r \in \{m, dtt, tt, k\text{-}tt, T, k\text{-}T, b\}$), let $R_r^p(\mathcal{C})$ denote the closure of $\mathcal{C}$ under r. That is, $R_r^p(\mathcal{C}) =_{df} \{L \mid (\exists L' \in \mathcal{C})[L \leq_r^p L']\}$. We refer the reader to any standard textbook in complexity theory (e.g. [BC93, HO02, Pap94]) for complexity classes and reductions not defined in this paper.

We introduce a notion called a $\Sigma_k(A)$-system. This notion is useful for concisely representing the computation of a stack of oracle NPTMs.

Definition 1. *1. For any $k \in \mathbb{N}^+$ and $A \subseteq \Sigma^*$, we call a tuple $[A; N_1, N_2, \ldots, N_k]$, where A is an oracle and $N_1, N_2, \ldots, N_k$ are oracle nondeterministic Turing machines, a $\Sigma_k(A)$-system. The* computation of a $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$ *on input x, denoted by $[A; N_1, N_2, \ldots, N_k](x)$, is defined as follows:*

- *For $k = 1$, $[A; N_1](x) =_{df} N_1^A(x)$, and*
- *for $k > 1$, $[A; N_1, N_2, \ldots, N_k](x) =_{df} N_1^{L(N_2^{\cdot^{\cdot^{L(N_k^A)}}})}(x)$.*

2. The language accepted by a $\Sigma_k(A)$-system*, denoted by $L[A; N_1, N_2, \ldots, N_k]$, is defined inductively as follows:*

$$L[A; N_1, N_2, \ldots, N_k] =_{df} \begin{cases} L(N_1^A) & \text{if } k = 1, \text{ and} \\ L(N_1^{L[A; N_2, N_3, \ldots, N_k]}) & \text{if } k > 1. \end{cases}$$

We capture the notion of unambiguity in $\Sigma_k(A)$-systems in the following definition.

Definition 2. *1. We say that a $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$ is* unambiguous *if for every $1 \le i \le k$ and for every $x \in \Sigma^*$, $[A; N_i, N_{i+1}, \ldots, N_k](x)$ has at most one accepting path.*

2. For any $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$, we define

$$L_{\text{unambiguous}}[A; N_1, N_2, \ldots, N_k] = \begin{cases} L[A; N_1, N_2, \ldots, N_k] & \text{if } [A; N_1, N_2, \ldots, N_k] \text{ is unambiguous}, \\ \text{undefined} & \text{otherwise}. \end{cases}$$

Roughly speaking, a property of an oracle machine is called *robust* if the machine retains that property with respect to every oracle. Below we define the property of robust unambiguity for a $\Sigma_k(A)$-system.

Definition 3. *We say that a $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$ is* robustly unambiguous *if for every set B, the $\Sigma_k(A \oplus B)$-system $[A \oplus B; N_1, N_2, \ldots, N_k]$ is unambiguous.*

2.2 Promise Problems and Smart Reductions

Even, Selman, and Yacobi [ESY84] introduced and studied the notion of promise problems. Promise problems are generalizations of decision problems in that the set of Yes-instances and the set of No-instances must partition the set of all instances in a decision problem, whereas this is not necessarily the case with promise problems. Over the years, the notion of promise problems has proved to be useful in complexity theory. (See [Gol05] for a nice survey on some such applications of promise problems.)

Definition 4 (Based on [Gol05]; cf. [ESY84]). *A promise problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ is defined in terms of disjoint sets $\Pi_{\text{yes}}, \Pi_{\text{no}} \subseteq \Sigma^*$. The set Π_{yes} is called the set of Yes-instances, the set Π_{no} is called the set of No-instances, and the set $\Pi_{\text{yes}} \cup \Pi_{\text{no}}$ is called the promise set.*

Definition 5. *A set L polynomial-time* smart *Turing reduces to a promise problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$, denoted by $L \le_{s,T}^p \Pi$ or $L \in \mathrm{P}_s^{\Pi}$, if there is a deterministic polynomial-time Turing machine M such that for all $x \in \Sigma^*$,*

1. *$x \in L \iff M^{\Pi}(x)$ accepts, and*
2. *if $M^{\Pi}(x)$ asks a query y to Π, then $y \in \Pi_{\text{yes}} \cup \Pi_{\text{no}}$.*

If on all input $x \in \Sigma^*$, the querying machine M asks at most k queries, for some constant $k \in \mathbb{N}^+$, then we say that L polynomial-time smart *k-Turing reduces to Π and write $L \leq^p_{s,k\text{-}T} \Pi$ or $L \in \mathrm{P}_s^{\Pi[k]}$.*

The following definitions are standard.

Definition 6. *Let Π be any promise problem. $R^p_{s,T}(\Pi)$ is the class of all sets L such that $L \leq^p_{s,T} \Pi$; for all $k \in \mathbb{N}^+$, $R^p_{s,k\text{-}T}(\Pi)$ is the class of all sets L such that $L \leq^p_{s,k\text{-}T} \Pi$; $R^p_{s,b}(\Pi)$ is the class of all sets L for which there exists some $k \in \mathbb{N}^+$ such that $L \leq^p_{s,k\text{-}T} \Pi$.*

Definition 7. *For any class of promise problems $\mathcal{C}$ and any reduction $r \in \{T, k\text{-}T, b\}$, we define $R^p_{s,r}(\mathcal{C}) =_{df} \bigcup_{\Pi \in \mathcal{C}} R^p_{s,r}(\Pi)$.*

We will study the computational power of smart Turing reductions to a particular class of promise problems, namely the class Promise-UP, which is defined as follows.

Definition 8. Promise-UP *is the class of all promise problems $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ for which there exists a nondeterministic polynomial-time Turing machine N such that for all $x \in \Sigma^*$, $x \in \Pi_{\text{yes}} \iff \#\mathrm{acc}_N(x) = 1,$ and $x \in \Pi_{\text{no}} \iff \#\mathrm{acc}_N(x) = 0.$*

The class $\mathrm{P}_s^{\text{Promise-UP}}$ of sets that polynomial-time smart Turing reduce to Promise-UP is a prominent class that behaves remarkably differently than the related class P^{UP}. While $\mathrm{P}_s^{\text{Promise-UP}}$ is known to contain the class FewP and the graph isomorphism problem [AK02], similar results for the case of P^{UP} are unknown.[1]

2.3 Unambiguity Based Hierarchies

Niedermeier and Rossmanith [NR98] observed that the notion of unambiguity in NPTMs can be generalized in three, perhaps distinct, ways to define unambiguity based hierarchies.

Definition 9 (Unambiguity Based Hierarchies [LR94, NR98]).

1. *The* alternating unambiguous polynomial hierarchy $\mathrm{AUPH} =_{df} \bigcup_{k \geq 0} \mathrm{AU}\Sigma^p_k = \bigcup_{k \geq 0} \mathrm{AU}\Pi^p_k$, *where* $\mathrm{AU}\Sigma^p_0 = \mathrm{AU}\Pi^p_0 =_{df} \mathrm{P}$ *and for every* $k \geq 1$, $\mathrm{AU}\Sigma^p_k = \exists! \cdot \mathrm{AU}\Pi^p_{k-1}$ *and* $\mathrm{AU}\Pi^p_k =_{df} \forall! \cdot \mathrm{AU}\Sigma^p_{k-1}$.[2]

[1] Arvind and Kurur [AK02] showed that the graph isomorphism problem (GI) belongs to the class SPP. Crasmaru et al. [CGRS04] observed that the proof of classifying GI into SPP, as given by Arvind and Kurur [AK02], actually yields a somewhat improved classification for GI. Their observation was that the graph isomorphism problem in fact belongs to $R^p_{s,T}(\text{Promise-UP})$, a subclass of SPP.

[2] For any arbitrary class $\mathcal{C}$, $\exists! \cdot \mathcal{C}$ is the class of all sets L for which there exists a polynomial $p(\cdot)$ and a set $L' \in \mathcal{C}$ such that for all $x \in \Sigma^*$, if $x \in L$ then there exists a unique $y \in \Sigma^{p(|x|)}$ such that $\langle x, y \rangle \in L'$, and if $x \notin L$ then for all $y \in \Sigma^{p(|x|)}$, $\langle x, y \rangle \notin L'$. Likewise, $\forall! \cdot \mathcal{C}$ is the class of all sets L for which there exists a polynomial $p(\cdot)$ and a set $L' \in \mathcal{C}$ such that for all $x \in \Sigma^*$, if $x \in L$ then for all $y \in \Sigma^{p(|x|)}$, $\langle x, y \rangle \in L'$, and if $x \notin L$ then there exists a unique $y \in \Sigma^{p(|x|)}$ such that $\langle x, y \rangle \notin L'$.

2. *The* unambiguous polynomial hierarchy *is* $\mathrm{UPH} =_{df} \bigcup_{k\geq 0} \mathrm{U}\Sigma_k^p$, *where* $\mathrm{U}\Sigma_0^p =_{df} \mathrm{P}$ *and for every* $k \geq 1$, $\mathrm{U}\Sigma_k^p =_{df} \mathrm{UP}^{\mathrm{U}\Sigma_{k-1}^p}$. *For each* $k \geq 0$, *the class* $\mathrm{U}\Pi_k^p =_{df} \mathrm{coU}\Sigma_k^p$.
3. *The* promise unambiguous polynomial hierarchy *is* $\mathcal{UPH} =_{df} \bigcup_{k\geq 0} \mathcal{U}\Sigma_k^p$, *where* $\mathcal{U}\Sigma_0^p =_{df} \mathrm{P}$, $\mathcal{U}\Sigma_1^p =_{df} \mathrm{UP}$, *and for every* $k \geq 2$, $\mathcal{U}\Sigma_k^p$ *is the class of all sets* $L \in \Sigma_k^p$ *such that for some oracle* NPTM*s* $N_1, N_2, \ldots, N_k$, $L = L(N_1^{L(N_2^{\cdot^{\cdot^{\cdot L(N_k)}}})})$, *and for every* $x \in \Sigma^*$ *and for every* $1 \leq i \leq k-1$, $N_1^{L(N_2^{\cdot^{\cdot^{\cdot L(N_k)}}})}(x)$ *has at most one accepting path and if* N_i *asks a query* w *to its oracle* $L(N_{i+1}^{\cdot^{\cdot^{\cdot L(N_k)}}})$ *during the computation of* $N_1^{\cdot^{\cdot^{\cdot L(N_k)}}}(x)$, *then* $N_{i+1}^{\cdot^{\cdot^{\cdot L(N_k)}}}(w)$ *has at most one accepting path. For each* $k \geq 0$, *the class* $\mathcal{U}\Pi_k^p =_{df} \mathrm{co}\mathcal{U}\Sigma_k^p$.

The following relationships among these complexity classes and other important classes are known.

Theorem 1. *1. For all* $k \geq 0$, $\mathrm{AU}\Sigma_k^p \subseteq \mathrm{U}\Sigma_k^p \subseteq \mathcal{U}\Sigma_k^p \subseteq \Sigma_k^p$ [LR94].
2. *For all* $k \geq 1$, $\mathrm{UP}_{\leq k} \subseteq \mathrm{AU}\Sigma_k^p \subseteq \mathrm{U}\Sigma_k^p \subseteq \mathcal{U}\Sigma_k^p \subseteq \mathrm{UAP} \subseteq \mathrm{SPP}$ ([LR94] + [NR98] + [CGRS04]).

3 Main Lemma

Our main lemma is Lemma 1, which we will use throughout this paper for generalizing known oracle constructions involving classes such as UP and Promise-UP to new oracle constructions involving arbitrary levels of the UPH. Roughly, Lemma 1 states the computational limitations of a $\Sigma_k(\mathcal{O})$-system, for any arbitrary $k \geq 1$, under certain weak conditions.

Lemma 1. *Fix a* $\Sigma_k(\mathcal{O})$*-system* $[\mathcal{O}; N_1, N_2, \ldots, N_k]$, *a string* $x \in \Sigma^*$, *and a set* $U \subseteq \Sigma^*$ *such that* $\mathcal{O} \cap U = \emptyset$. *Let* $r(.)$ *be a polynomial that bounds the running time of each of the machines* $N_1, N_2, \ldots, N_k$. *Then the following holds:*

1. *Suppose* $[\mathcal{O}; N_1, N_2, \ldots, N_k](x)$ *accepts and for every* $A \subseteq U$ *with* $||A|| \leq k$, $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_k]$ *is unambiguous. Let*
$$C = \{\alpha \in U \mid [\mathcal{O} \cup \{\alpha\}; N_1, N_2, \ldots, N_k](x) \text{ rejects}\}.$$
Then $||C|| \leq 5^k \cdot \prod_{i=1}^k (r\circ)^i(|x|)$.
2. *Suppose* $[\mathcal{O}; N_1, N_2, \ldots, N_k](x)$ *rejects and for every* $A \subseteq U$ *with* $||A|| \leq k+1$, $[\mathcal{O} \cup A; N_1, N_2, \ldots, N_k]$ *is unambiguous. Let*
$$C = \{\alpha \in U \mid [\mathcal{O} \cup \{\alpha\}; N_1, N_2, \ldots, N_k](x) \text{ accepts}\}.$$
Then $||C|| \leq 5^k \cdot \prod_{i=1}^k (r\circ)^i(|x|)$.

Any oracle machine can be interpreted as a function mapping a set of strings to another set of strings as follows: A machine N maps any set $\mathcal{O}$ to the set $L(N^{\mathcal{O}})$. Therefore it makes sense to consider the function $\mathcal{L} : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$ defined by a $\Sigma_k(\cdot)$-system $[\cdot; N_1, N_2, \ldots, N_k]$. (That is, define $\mathcal{L}$ so that for every $\mathcal{O} \subseteq \Sigma^*$, $\mathcal{L}(\mathcal{O}) =_{df} L[\mathcal{O}; N_1, N_2, \ldots, N_k]$.) We introduce a convenient notion called "(h,t)-ambiguity" for (partial) functions such as the ones defined by $\Sigma_k(\cdot)$-systems.

Definition 10. *For any $h \in \mathbb{N}^+$ and polynomial $t(\cdot)$, we call a partial function $\mathcal{L} : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$* (h,t)-ambiguous *if for every $\mathcal{O}, U \subseteq \Sigma^*$ with $\mathcal{O} \cap U = \emptyset$, one of the following is true:*

1. *For some $A \subseteq U$ with $||A|| \leq h$, $\mathcal{L}(\mathcal{O} \cup A)$ is undefined, or*
2. *for every $w \in \Sigma^*$,*

$$||\{\alpha \in U \mid w \in \mathcal{L}(\mathcal{O} \cup \{\alpha\}) \Longleftrightarrow w \notin \mathcal{L}(\mathcal{O})\}|| \leq t(|w|).$$

The machine N_1 in a $\Sigma_k(\mathcal{O})$-system $[\mathcal{O}; N_1, N_2, \ldots, N_k]$ has oracle access to the set $L[\mathcal{O}; N_2, N_3, \ldots, N_k]$. In many of our proofs, we first apply Lemma 1 to prove that under certain conditions the $\Sigma_{k-1}(\cdot)$-subsystem $[\cdot; N_2, N_3, \ldots, N_k]$ defines a (k,t)-ambiguous function $\mathcal{L}'$, where t is some polynomial and for any $\mathcal{O} \subseteq \Sigma^*$, $\mathcal{L}'(\mathcal{O})$ is defined to be $L[\mathcal{O}; N_2, N_3, \ldots, N_k]$. Then we can assume that the machine N_1 has oracle access to the set $\mathcal{L}(\mathcal{O})$, where $\mathcal{L}$ can be any arbitrary (k,t)-ambiguous function, rather than to the set $L[\mathcal{O}; N_2, N_3, \ldots, N_k]$. This works because the (k,t)-ambiguity of the function $\mathcal{L}'$ defined by the $\Sigma_{k-1}(\cdot)$-subsystem $[\cdot; N_2, N_3, \ldots, N_k]$ is the only property of $\mathcal{L}'$ that is needed in the proofs. This approach greatly simplifies our proof arguments since we will not need to deal with stacks of oracle NPTMs.

4 Applications

4.1 Comparing Bounded Ambiguity Classes with the Levels of UPH

We compare nondeterministic polynomial-time complexity classes ($\mathrm{UP}_{O(1)}$ and FewP), which are based on Turing machines having restrictions on the number of accepting paths, with levels of the unambiguous polynomial hierarchy (UPH). It is known that $\mathrm{UP}_{\leq k} \subseteq \mathrm{U}\Sigma_k^p$ in all relativized worlds. Theorem 2 shows the optimality of this inclusion with respect to relativizable proof techniques. Beigel [Bei89] constructed an oracle relative to which $\mathrm{UP}_{k(n)+1} \not\subseteq \mathrm{UP}_{k(n)}$, for every polynomial $k(n) \geq 2$. Theorem 2 subsumes this oracle result of Beigel [Bei89] for any constant k.

By a slight modification of the oracle construction in Theorem 2, we can show that the second level of the promise unambiguous hierarchy $\mathcal{U}\Sigma_2^p$ is not contained in the unambiguous polynomial hierarchy UPH. Results on relativized separations of levels of some unambiguity based hierarchy from another hierarchy have been investigated earlier. Rossmanith (see [NR98]) gave a relativized separation of $\mathrm{AU}\Sigma_k^p$ from $\mathrm{U}\Sigma_k^p$, for any $k \geq 2$. Spakowski and Tripathi [ST] constructed an oracle relative to which $\mathrm{AU}\Sigma_k^p \not\subseteq \Pi_k^p$, for any $k \geq 1$. Our relativized separation of $\mathcal{U}\Sigma_2^p$ from UPH does not seem to be implied from these previous results in any obvious way.

Theorem 2. $(\forall k \geq 1)(\exists \mathcal{A})[\mathrm{UP}^{\mathcal{A}}_{\leq k+1} \not\subseteq \mathrm{U}\Sigma^{p,\mathcal{A}}_{k}]$.

A straightforward adaptation of the proof technique in Theorem 2 allows to separate the second level, $\mathcal{U}\Sigma^p_2$, of the promise unambiguous polynomial hierarchy from the unambiguous polynomial hierarchy, UPH, in some relativized world. We obtain this relativized separation via Theorem 3, where a subclass, namely $\mathrm{FewP}^{\mathcal{A}}$, of $\mathcal{U}\Sigma^{p,\mathcal{A}}_2$ is separated from $\mathrm{UPH}^{\mathcal{A}}$.

Theorem 3. $(\exists \mathcal{A})\left[\mathrm{FewP}^{\mathcal{A}} \not\subseteq \mathrm{UPH}^{\mathcal{A}}\right]$.

Corollary 1. *There is a relativized world where $\mathcal{U}\Sigma^p_2$ is not contained in* UPH.

Cai, Hemachandra, and Vyskoč [CHV93] proved that smart 2-Turing access to Promise-UP cannot be subsumed by $\mathrm{coNP}^{\mathrm{UP}} \cup \mathrm{NP}^{\mathrm{UP}}$ in some relativized world. As a consequence, they showed that there is a relativized world where smart bounded adaptive reductions to Promise-UP and smart nonadaptive reductions to Promise-UP are nonequivalent, a characteristic that stands in contrast with the cases of UP and NP. (Both UP and NP are known to have equivalence between bounded adaptive reductions and nonadaptive reductions in all relativized worlds (see [CHV93, Wag90].) We generalize their result in Theorem 4, where we prove that smart k-Turing access to Promise-UP cannot be relativizably contained in $\mathrm{coNP}^{\mathrm{U}\Sigma^{p,A}_{k-1}} \cup \mathrm{NP}^{\mathrm{U}\Sigma^{p,A}_{k-1}}$, for any $k \geq 2$.

Theorem 4. $(\forall k \geq 2)(\exists \mathcal{A})\left[R^p_{s,k\text{-}T}(\text{Promise-UP}^{\mathcal{A}}) \not\subseteq \mathrm{coNP}^{\mathrm{U}\Sigma^{p,A}_{k-1}} \cup \mathrm{NP}^{\mathrm{U}\Sigma^{p,A}_{k-1}}\right]$.

4.2 Simulating Nonadaptive Access by Adaptive Access (Non-promise Case)

It is known that adaptive Turing access to NP is exponentially more powerful compared to nonadaptive Turing access to NP. That is, $R^p_{(2^k-1)\text{-}tt}(\mathrm{NP}) \subseteq R^p_{k\text{-}T}(\mathrm{NP})$ [Bei91] and this inclusion relativizes. However, for the case of unambiguous nondeterministic computation such a relationship between nonadaptive access and adaptive access is not known. Cai, Hemachandra, and Vyskoč [CHV92] showed that even proving the superiority of adaptive Turing access over nonadaptive Turing access with one single query more might be nontrivial for unambiguous nondeterministic computation:

Theorem 5 ([CHV92]). *For any total, polynomial-time computable and polynomially bounded function $k(\cdot)$, there exists an oracle $\mathcal{A}$ such that*

$$R^p_{(k(n)+1)\text{-}tt}(\mathrm{UP}^{\mathcal{A}}) \not\subseteq R^{p,\mathcal{A}}_{k(n)\text{-}T}(\mathrm{UP}^{\mathcal{A}}).$$

In the next theorem, we generalize this result to the higher levels of the unambiguous polynomial hierarchy UPH.

Theorem 6. *For any total, polynomial-time computable and polynomially bounded function $k(\cdot)$, and $h \in \mathbb{N}^+$, there exists an oracle $\mathcal{A}$ such that*

$$\mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h}) \not\subseteq \mathrm{R}^{p,\mathcal{A}}_{k(n)\text{-}T}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h),$$

and hence $\mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h) \not\subseteq \mathrm{R}^{p,\mathcal{A}}_{k(n)\text{-}T}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h)$.

4.3 Simulating Nonadaptive Access by Adaptive Access (Promise Case)

Cai, Hemachandra, and Vyskoč [CHV93] proved the following partial improvement of their Theorem 5.

Theorem 7 ([CHV93]). *For any constant k, there exists an oracle $\mathcal{A}$ such that*

$$R^p_{(k+1)\text{-}tt}(\mathrm{UP}^{\mathcal{A}}) \not\subseteq R^{p,\mathcal{A}}_{s,k\text{-}T}(\text{Promise-UP}^{\mathcal{A}}).$$

Note that we have replaced "UP" by "Promise-UP" on the righthand side of the non-inclusion relation of Theorem 5. This is a significant improvement for the following reason. The computational powers of $R^p_b(\mathrm{UP})$ and $R^p_{s,b}(\text{Promise-UP})$ (the bounded Turing closure of UP and the bounded smart Turing closure of Promise-UP, respectively) are known to be remarkably different in certain relativized worlds. While it is easy to show that $\mathrm{UP}_{\leq k}$ is robustly (i.e., for every oracle) contained in $\mathrm{P}^{\text{Promise-UP}}_{s,k\text{-}T}$ for any $k \geq 1$, we have shown in Theorem 2 that for no $k \geq 2$, $\mathrm{UP}_{\leq k}$ is robustly contained in P^{UP}. Therefore, it is not immediately clear whether this improvement is impossible, i.e. whether $R^p_{(k+1)\text{-}tt}(\mathrm{UP}) \subseteq R^p_{s,k\text{-}T}(\text{Promise-UP})$ holds relative to all oracles.

However, Cai, Hemachandra, and Vyskoč [CHV93] could achieve this improvement only by paying a heavy price. In their own words:

> In our earlier version dealing with UP^A, the constant k can be replaced by any arbitrary polynomial-time computable function $f(n)$ with polynomially bounded value. It remains open whether the claim of the current strong version of Theorem 3.1 can be similarly generalized to non-constant access.

We resolve this open question. We show that Theorem 7 holds with constant k replaced by any total, polynomial-time computable and polynomially bounded function $k(\cdot)$. This result is subsumed as the special case $h = 1$ of our main result, Theorem 8, of this subsection.

Theorem 8. *For any total, polynomial-time computable and polynomially bounded function $k(\cdot)$, and $h \in \mathbb{N}^+$, there exists an oracle $\mathcal{A}$ such that*

$$\mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{UP}^{\mathcal{A}}_{\leq h}) \not\subseteq \mathrm{R}^{p,\mathcal{A}}_{s,k(n)\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}}),$$

and hence $\mathrm{R}^p_{(k(n)+1)\text{-}dtt}(\mathrm{U}\Sigma^{p,\mathcal{A}}_h) \not\subseteq \mathrm{R}^{p,\mathcal{A}}_{s,k(n)\text{-}T}(\text{Promise-UP}^{\mathrm{U}\Sigma^{p,\mathcal{A}}_{h-1}})$.

Theorem 8 is furthermore a generalization of Theorem 7 to higher levels of the unambiguous polynomial hierarchy.

4.4 Simulating Adaptive Access by Nonadaptive Access

Sections 4.2 and 4.3 studied the limitations of simulating nonadaptive queries to $\mathrm{UP}_{\leq h}$ by adaptive queries to $\mathrm{U}\Sigma^p_h$ in relativized settings. This section complements these investigations. In particular, Theorem 9 of this section shows that in a certain relativized world, it is impossible to simulate adaptive k-Turing access to $\mathrm{UP}_{\leq h}$ by nonadaptive (2^k-2)-tt access to $\mathrm{U}\Sigma^p_h$. This also implies optimality of robustly (i.e., for every oracle) simulating adaptive k-Turing accesses by nonadaptive $(2^k - 1)$-tt accesses to classes such as $\mathrm{UP}_{\leq h}$ and $\mathrm{U}\Sigma_h^p$, since for any class $\mathcal{C}$, we can trivially, via brute-force method, simulate adaptive k-Turing reduction to the class by nonadaptive $(2^k - 1)$-tt reduction to the same class.

Theorem 9 generalizes a result of Cai, Hemachandra, and Vyskoč [CHV93] from the case of $h = 1$ to the case of arbitrary constant $h \geq 1$.

Theorem 9. *For any constants $k, h \in \mathbb{N}^+$, there exists an oracle $\mathcal{A}$ such that*

$$R_{k\text{-}T}^p(\mathrm{UP}_{\leq h}^{\mathcal{A}}) \not\subseteq R_{(2^k-2)\text{-}tt}^{p,\mathcal{A}}(\mathrm{NP}^{\mathrm{U}\Sigma_{h-1}^{p,\mathcal{A}}}),$$

and hence $R_{k\text{-}T}^p(\mathrm{UP}_{\leq h}^{\mathcal{A}}) \not\subseteq R_{(2^k-2)\text{-}tt}^{p,\mathcal{A}}(\mathrm{U}\Sigma_h^{p,\mathcal{A}})$.

4.5 Fault-Tolerant Access

Ko [Ko87] introduced the notion of *one-sided helping* by a set A in the computation of a set B. A set A is said to provide *one-sided help* to a set B if there is a deterministic oracle Turing machine M computing B and a polynomial $p(\cdot)$ such that (a) on any input $x \in B$, $M^A(x)$ accepts in time $p(|x|)$, and (b) for all inputs y and for all oracles C, $M^C(y)$ accepts (though perhaps $M^C(y)$ may take a longer time than $p(|y|)$) if and only if $y \in B$. Since the machine M, accepting the set B, is capable of answering correctly on faulty oracles, i.e. oracles C different from the oracle A that provides one-sided help to B, the oracle access mechanism is termed fault-tolerant (see [CHV93]). Ko [Ko87] defined $\mathrm{P}_{1\text{-help}}(A)$ to be the class of all sets B that can be one-sided helped by A.

We generalize and improve the relativized separation of $\mathrm{P}_{1\text{-help}}(\mathrm{UP})$ from UP by Cai, Hemachandra, and Vyskoč [CHV93] in Theorem 10.

Theorem 10. *For all $h \geq 1$, there exists an oracle $\mathcal{A}$ such that*

$$\mathrm{P}_{1\text{-help}}(\mathrm{UP}_{\leq h}^{\mathcal{A}}) \not\subseteq R_{s,b}^{p,\mathcal{A}}(\text{Promise-UP}^{\mathrm{U}\Sigma_{h-1}^{p,\mathcal{A}}}).$$

5 Robust Unambiguity

So far we looked at several applications of Lemma 1 in constructing relativized worlds involving arbitrary levels of the unambiguous polynomial hierarchy. Lemma 1, in essence, shows the computational limitations of a $\Sigma_k(A)$-system under certain weak restrictions. What if we impose a more stringent restriction on a $\Sigma_k(A)$-system? This question is relevant to our next investigation.

We study the power of robustly unambiguous $\Sigma_k(A)$-system in Theorem 11. (Recall from Section 2, a $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$ is robustly unambiguous if for every oracle B, $[A \oplus B; N_1, N_2, \ldots, N_k]$ is unambiguous.) Theorem 11 illustrates the following fact: A robustly unambiguous $\Sigma_k(A)$-system is so weak that given any oracle set B and input x, the hierarchical nondeterministic polynomial-time oracle access to B in $[A \oplus B; N_1, N_2, \ldots, N_k](x)$ can be stripped down and turned into a deterministic polynomial-time oracle access (to B) without changing the acceptance behavior of the $\Sigma_k(A \oplus B)$-system on input x. As a corollary, we obtain a generic collapse of $\mathrm{U}\Sigma_k^p$ to P, for each $k \geq 1$, assuming $\mathrm{P} = \mathrm{NP}$.

Theorem 11. *For all $A \subseteq \Sigma^*$ and $k \geq 1$, if the $\Sigma_k(A)$-system $[A; N_1, N_2, \ldots, N_k]$ is robustly unambiguous, then for every $B \subseteq \Sigma^*$,*

$$L[A \oplus B; N_1, N_2, \ldots, N_k] \in \mathrm{P}^{\Sigma_k^{p,A} \oplus B}.$$

Corollary 2. *If* $\mathrm{P} = \mathrm{NP}$*, then relative to a (Cohen) generic G,* $\mathrm{P} = \mathrm{U}\Sigma_k^p$ *for all $k \geq 1$.*

The last corollary generalizes a result of Blum and Impagliazzo: If $\mathrm{P} = \mathrm{NP}$, then relative to a (Cohen) generic G, $\mathrm{P}^G = \mathrm{UP}^G$ [BI87]. Fortnow and Yamakami [FY96] demonstrated that similar collapses relative to any (Cohen) generic G do not occur at higher levels of the polynomial hierarchy. They proved that for each $k \geq 2$, there exists a tally set in $\mathrm{UP}^{\Sigma_{k-1}^{p,G}, G} \cap \Pi_k^{p,G}$ but not in $\mathrm{P}^{\Sigma_{k-1}^{p,G}, G}$. Thus Corollary 2 contrasts with this generic separation by Fortnow and Yamakami.

Acknowledgment. We thank Lane Hemaspaandra and Jörg Rothe for their constant encouragement and support.

References

[ACRW04] S. Aida, M. Crâsmaru, K. Regan, and O. Watanabe. Games with uniqueness properties. *Theory of Computing Systems*, 37(1):29–47, 2004.

[AK02] V. Arvind and P. Kurur. Graph isomorphism is in SPP. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 743–750, Los Alamitos, November 2002. IEEE Computer Society.

[BC93] D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1993.

[Bei89] R. Beigel. On the relativized power of additional accepting paths. In *Proceedings of the 4th Structure in Complexity Theory Conference*, pages 216–224. IEEE Computer Society Press, June 1989.

[Bei91] R. Beigel. Bounded queries to SAT and the boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, 1991.

[Bei93] R. Beigel. The polynomial method in circuit complexity. In *Proceedings of the 8th Structure in Complexity Theory Conference*, pages 82–95, San Diego, CA, USA, May 1993. IEEE Computer Society Press.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[BI87] M. Blum and R. Impagliazzo. Generic oracles and oracle classes. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 118–126, October 1987.

[BS79] T. Baker and A. Selman. A second step toward the polynomial hierarchy. *Theoretical Computer Science*, 8:177–187, 1979.

[BU98] C. Berg and S. Ulfberg. A lower bound for perceptrons and an oracle separation of the $\mathrm{PP}^{\mathrm{PH}}$ hierarchy. *Journal of Computer and System Sciences*, 56(3):263–271, 1998.

[CGRS04] M. Crâsmaru, C. Glaßer, K. Regan, and S. Sengupta. A protocol for serializing unique strategies. In *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science*. Springer-Verlag *Lecture Notes in Computer Science #3153*, August 2004.

[CHV92] J. Cai, L. Hemachandra, and J. Vyskoč. Promise problems and access to unambiguous computation. In *Proceedings of the 17th Symposium on Mathematical Foundations of Computer Science*, pages 162–171. Springer-Verlag *Lecture Notes in Computer Science #629*, August 1992.

[CHV93] J. Cai, L. Hemachandra, and J. Vyskoč. Promises and fault-tolerant database access. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory*, pages 101–146. Cambridge University Press, 1993.

[ESY84] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[FY96] L. Fortnow and T. Yamakami. Generic separations. *Journal of Computer and System Sciences*, 52(1):191–197, February 1996.

[Gol05] O. Goldreich. On promise problems. Technical report TR05–018, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.

[GT05] C. Glaßer and S. Travers. Machines that can output empty words. Technical report TR05–147, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[Hås87] J. Håstad. *Computational Limitations of Small-Depth Circuits*. MIT Press, 1987.

[HO02] L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. Springer, 2002.

[Ko85] K. Ko. On some natural complete operators. *Theoretical Computer Science*, 37(1):1–30, 1985.

[Ko87] K. Ko. On helping by robust oracle machines. *Theoretical Computer Science*, 52:15–36, 1987.

[Ko89] K. Ko. Relativized polynomial time hierarchies having exactly k levels. *SIAM Journal on Computing*, 18(2):392–408, 1989.

[LR94] K.-J. Lange and P. Rossmanith. Unambiguous polynomial hierarchies and exponential size. In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 106–115. IEEE Computer Society Press, June/July 1994.

[NR98] R. Niedermeier and P. Rossmanith. Unambiguous computations and locally definable acceptance types. *Theoretical Computer Science*, 194(1–2):137–161, 1998.

[OH93] M. Ogiwara and L. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Reg97] K. Regan. Polynomials and combinatorial definitions of languages. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 261–293. Springer-Verlag, 1997.

[Sip83] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 61–69. ACM Press, 1983.

[SL96] M. Sheu and T. Long. UP and the low and high hierarchies: A relativized separation. *Mathematical Systems Theory*, 29(5):423–449, 1996.

[ST] H. Spakowski and R. Tripathi. On the power of unambiguity in alternating machines. *Theory of Computing Systems*. To appear.

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

[Yao85] A. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.

An Efficient Algorithm Finds Noticeable Trends and Examples Concerning the Černy Conjecture

A.N. Trahtman

Bar-Ilan University, Dep. of Math., 52900, Ramat Gan, Israel
trakht@macs.biu.ac.il
http://www.cs.biu.ac.il/∼trakht/Testas.html

Abstract. A word w is called synchronizing (recurrent, reset, directed) word of a deterministic finite automaton (DFA) if w sends all states of the automaton on a unique state. Jan Černy had found in 1964 a sequence of n-state complete DFA with shortest synchronizing word of length $(n-1)^2$. He had conjectured that it is an upper bound for the length of the shortest synchronizing word for any n-state complete DFA.

The examples of DFA with shortest synchronizing word of length $(n-1)^2$ are relatively rare. To the Černy sequence were added in all examples of Černy, Piricka and Rosenauerova (1971), of Kari (2001) and of Roman (2004).

By help of a program based on some effective algorithms, a wide class of automata of size less than 11 was checked. The order of the algorithm finding synchronizing word is quadratic for overwhelming majority of known to date automata. Some new examples of n-state DFA with minimal synchronizing word of length $(n-1)^2$ were discovered. The program recognized some remarkable trends concerning the length of the minimal synchronizing word.

Keywords: Deterministic finite automaton, synchronizing word, algorithm, complexity, Černy conjecture.

Introduction

We consider a DFA with complete state transition graph Γ and transition semigroup S over alphabet Σ. Let n be the size of DFA and q be the size of Σ.

The problem of synchronization of DFA is natural and various aspects of this problem were touched upon the literature. Synchronization makes the behavior of an automaton resistant against input errors since, after detection of an error, a synchronizing word can reset the automaton back to its original state, as if no error had occurred. Therefore different problems of synchronization draw the attention.

A problem with a long story is the estimation of the minimal length of synchronizing word. Most known as a Černy conjecture, it was aroused independently by distinct authors. Jan Černy had found in 1964 [2] n-state complete DFA with shortest synchronizing word of length $(n-1)^2$ for $q=2$. He had conjectured that it is an upper bound for the length of the shortest synchronizing word for any

n-state complete DFA. The problem can be reduced to automata with strongly connected graph [2]. The best known upper bound is now equal to $(n^3 - n)/6$ [5], [8], [9], [12]. The conjecture holds true for a lot of automata, but in general the problem remains open. This simply looking conjecture is now one of the most longstanding open problems in the theory of finite automata. Moreover, the examples of automata with shortest synchronizing word of length $(n - 1)^2$ are infrequent. After the sequence found by Černy and example of Černy, Piricka and Rosenauerova [3] of 1971 for $q = 2$, the next such example was found by Kari [6] only in 2001 for $n = 6$ and $q = 2$. Roman [14] had found an analogical example for $n = 5$ and $q = 3$ in 2004. There are no examples of automata for the time being such that the length of the shortest synchronizing word is greater than $(n - 1)^2$.

The testing of synchronizing automata is an indispensable part of investigation in this area [1], [4], [10], [11], [13], [19]. The best known to date algorithm of Eppstein [4], [10] improves an algorithm of Natarjan [11] and finds a synchronizing word for n-state DFA in $O(n^3 + n^2q)$ time.

We present a new efficient algorithm for finding a synchronizing word. The actual running time of the algorithm on a lot of examples proved to be essentially less than in case of $O(n^3q)$ time complexity. For clear majority of automata, the time complexity is $O(n^2q)$. It gives a chance to extend noticeably the class of considered DFA. This algorithm plays a central role in the program for search of automata with minimal reset word.

The program studied all automata with strongly connected transition graph of size $n \leq 10$ for $q = 2$ and of size $n \leq 7$ for $q \leq 4$. All known and some new examples of DFA with shortest synchronizing word of length $(n - 1)^2$ from this class of automata were checked. So all examples of DFA with shortest synchronizing word of length $(n - 1)^2$ in this area are known for today. The size of the alphabet of the examples is two or three. The situation in the neighborhood of the bound $(n-1)^2$ of Černy (minimal reset words of relatively great length) was also studied.

There are no contradictory examples for the Černy conjecture in this class of automata. Moreover, the program does not find new examples of DFA with reset word of length $(n - 1)^2$ for automata with $n > 4$ as well as for $q > 3$. No such examples exist for alphabet of size four if $n \leq 7$.

And what is more, the examples with minimal length of reset word disappear even for values near the Černy bound $(n - 1)^2$ with growth of the size of the automaton as well as of the size of the alphabet. The gap between $(n - 1)^2$ and the nearest of the minimal lengths of reset word appears for $n = 6$. There are no 6-state automata with minimal length of synchronizing word of 24 for $q \leq 4$.

The following table displays this interesting trend for the length of minimal reset words less than $(n - 1)^2$.

size	n=5 $q <= 4$	n=6 $q <= 4$	n=7 $q <= 4$	n=8 q=2	n=9 q=2	n=10 q=2
$(n-1)^2$	16	25	36	49	64	81
max length	15	23	32	44	58	74

The program uses also straightforward algorithm for finding synchronizing word of minimal length. A help algorithm of the program verifies whether or not a given DFA is synchronizing. It is a modification of an algorithm of $O(n^2q)$ time complexity supposed by Eppstein [4], [10]. Our version has $O(n^2q)$ time complexity only in the worst case and we use usually only its linear part.

The comparison of the experimental data suggests that the length of the synchronizing word found by central algorithm of the program is not far from the length of the minimal synchronizing word. This length was not greater than n^2 in all billions cases studied for today. The results of the algorithms altogether correspond to the Černy conjecture. All above algorithms are implemented in our package TESTAS [19].

Preliminaries

Let us consider a deterministic finite automaton with state transition graph Γ and transition semigroup S over alphabet Σ. The states of the automaton are considered below as vertices of the transition graph Γ.

The number of vertices of the graph Γ is denoted by $|\Gamma|$.

A maximal strongly connected component of a directed graph will be denoted for brevity as **SCC**.

If there exists a path $v \in \Sigma^+$ from vertex $\mathbf{p}$ to vertex $\mathbf{q}$ in the transition graph of DFA then let us denote the vertex $\mathbf{q}$ as $\mathbf{p}v$.

Let Γv denote the mapping of the graph [automaton] Γ by help of $v \in \Sigma^+$, let us call $|\Gamma v|$ *rank* and $|\Gamma| - |\Gamma v|$ *defect* of the mapping v.

A word $v \in \Sigma^+$ is called *synchronizing word* of an automaton A with transition graph Γ if $|\Gamma v| = 1$. An automaton (and its transition graph) possessing a synchronizing word is called *synchronizing.*

A word w is called *2-reset word* of the pair $\mathbf{p}, \mathbf{q}$ if $\mathbf{p}w = \mathbf{q}w$.

Suppose $\mathbf{p} \succeq \mathbf{q}$ if $\mathbf{p}w = \mathbf{q}$ for some word w.

A state [a vertex] $\mathbf{q}$ is called *sink* of an automaton [of a graph] if $\mathbf{p} \succeq \mathbf{q}$ for all $\mathbf{p}$.

An automaton [a graph Γ] is called *complete* if for every state [vertex] $\mathbf{p}$ and every $\sigma \in \Sigma$ the state [vertex] $\mathbf{p}\sigma$ exists.

The direct product Γ^2 of two copies of graph Γ over an alphabet Σ consists of vertices $(\mathbf{p}, \mathbf{q})$ and edges $(\mathbf{p}, \mathbf{q}) \to (\mathbf{p}\sigma, \mathbf{q}\sigma)$ labelled by σ. Here $\mathbf{p}, \mathbf{q} \in \Gamma$, $\sigma \in \Sigma$.

1 Some Auxiliary Properties

Two following two simple lemmas belong rather to the folklore.

Lemma 1. *[2] [18], [10] The directed labelled graph Γ is synchronizing if and only if Γ^2 has sink state.*

Lemma 2. *[18] The sets of synchronizing words of the graphs Γ and Γ^2 coincide.*

Lemma 3. *Let Γ be strongly connected graph of synchronizing automaton. Then for every state $\mathbf{p} \in \Gamma$ there exists a word s of length not greater than $|\Gamma|$ such that $\mathbf{p} \notin \Gamma s$.*

Proof. Let us denote the set of states from $\Gamma \setminus \Gamma u$ for all u such that $|u| \leq i$ as Q_i. The complement of the set Q_i let us denote as C_i.

The automaton is synchronizing. Therefore there exists a letter α such that $|\Gamma\alpha| < |\Gamma|$, whence Q_1 is not empty. The graph Γ is strongly connected. Therefore there exists a letter β and a state $\mathbf{c} \in C_{i-1}$ such that $\mathbf{c}\beta \in Q_{i-1}$. Hence $|C_{i-1}\beta \cap C_{i-1}| < |C_{i-1}|$ and the states from $C_{i-1}\beta \setminus C_{i-1}$ belong to Q_i. Thus $Q_{i-1} \subset Q_i$ and the size of Q_i is growing with i.

Consequently, for some $i \leq |\Gamma|$ $|Q_i| = |\Gamma|$. Thus for every state $\mathbf{p} \in \Gamma$ there exists a word s of length not greater than $|\Gamma|$ such that $\mathbf{p} \in Q_{|s|}$ and $\mathbf{p} \notin \Gamma s$.

Lemma 4. *Suppose $\mathbf{p} \notin \Gamma s$ for a word s and a state $\mathbf{p}$ of transition graph Γ of DFA.*

Then there exist two minimal integer k and r such that $\mathbf{p}s^k = \mathbf{p}s^{k+r}$. The pair of states $\mathbf{p}, \mathbf{p}s^r$ has 2-reset word s^k and for every $i < k$ the pair of states $\mathbf{p}s^i, \mathbf{p}s^{r+i}$ has 2-reset word s^{k-i}. The word s^k is a 2-reset word for at least k different pairs of states.

In the case $r = 1$ every pair of states $\mathbf{p}s^i, \mathbf{p}s^k$ for every $i < k$ has 2-reset word s^{k-i}.

Proof. The sequence $\mathbf{p}s, \mathbf{p}s^2, ..., \mathbf{p}s^t, ...$ is finite and belongs to Γs. Therefore such k and r exist. Two states $\mathbf{p}s^i$ and $\mathbf{p}s^{r+i}$ are mapped by the power s^{k-i} on $\mathbf{p}s^k = \mathbf{p}s^{k+r}$ as well as the states $\mathbf{p}$ and $\mathbf{p}s^r$ are mapped by the power s^k on $\mathbf{p}s^k$. All states $\mathbf{p}s^i$ are distinct for $i \leq k$, whence the word s^k unites at least k distinct pairs of states.

In the case $r = 1$, two states $\mathbf{p}s^i$ and $\mathbf{p}s^k$ are mapped by the word s^{k-i} on $\mathbf{p}s^k = \mathbf{p}s^{k+1}$ as well as the pair of states $\mathbf{p}$, $\mathbf{p}s^k$ is mapped by the power s^k on $\mathbf{p}s^k$. All states $\mathbf{p}s^i$ are distinct for $i \leq k$, whence the word s^k unites also in this case at least k distinct pairs of states.

Lemma 5. *Suppose $\mathbf{r}\alpha = \mathbf{t}\alpha$ for a letter α and two distinct states $\mathbf{r}$, $\mathbf{t}$ of transition graph Γ of DFA and let the states $\mathbf{r}$ and $\mathbf{r}\alpha$ be consecutive states of a cycle C of Γ.*

Then there exists a word s of length of the cycle C such that $\mathbf{r}s = \mathbf{r}$ and $|\Gamma s| < |\Gamma|$. For some state $\mathbf{p} \in \Gamma \setminus \Gamma s$ there exists a minimal integer k such that $\mathbf{p}s^k = \mathbf{p}s^{k+1}$. The pair of states $\mathbf{p}, \mathbf{p}s^k$ has 2-reset word s^k and for every $i < k$ the pair of states $\mathbf{p}s^i, \mathbf{p}s^k$ has 2-reset word s^{k-i}. The word s^k unites at least $k+1$ distinct states.

Proof. A word s with first letter α can be obtained from consecutive letters on the edges of the cycle C. Therefore $|s|$ is equal to the length of the cycle and $\mathbf{r}s = \mathbf{r}$. $|\Gamma s| < |\Gamma|$ follows from $\mathbf{r}\alpha = \mathbf{t}\alpha$.

From $\mathbf{r}s = \mathbf{r} \neq \mathbf{t}$ and $\mathbf{r}\alpha = \mathbf{t}\alpha$ follows that $\mathbf{t}s = \mathbf{r} \neq \mathbf{t}$ and $\mathbf{t}s^i \neq \mathbf{t}$ for any integer i. In the case $\mathbf{t} \in \Gamma \setminus \Gamma s$ suppose $\mathbf{p} = \mathbf{t}$, and so the state $\mathbf{p}$ is defined. In opposite case for some state $\mathbf{t}_1$ holds $\mathbf{t}_1 s = \mathbf{t}$. If $\mathbf{t}_1 \in \Gamma \setminus \Gamma s$ suppose

$\mathbf{p} = \mathbf{t}_1$, else for some state $\mathbf{t}_2$ holds $\mathbf{t}_2 s^2 = \mathbf{t}$. Let us continue this procedure until $\mathbf{t}_{k-1} \in \Gamma \setminus \Gamma s$ for some k such that $\mathbf{t}_{k-1} s^{k-1} = \mathbf{t}$. Such minimal k exists and all states $\mathbf{t}$, $\mathbf{t}_1$, ..., $\mathbf{t}_j$ for $j \leq k$ are distinct because $\mathbf{t}s^i \neq \mathbf{t}$ for any integer i. The state $\mathbf{t}$ therefore has a preimage $\mathbf{p} = \mathbf{t}_{k-1}$ in $\Gamma \setminus \Gamma s$ by mapping s^{k-1}, whence $\mathbf{p}s^k = \mathbf{p}s^{k+1} = \mathbf{r}$.

So the pair of states $\mathbf{p}, \mathbf{p}s^k$ has 2-reset word s^k and for every $i < k$ the pair of states $\mathbf{p}s^i, \mathbf{p}s^k$ has 2-reset word s^{k-i}. The states $\mathbf{p}s^i$ for $i \leq k$ and $\mathbf{p}$ are distinct because of the choice of k. The word s^k maps all these states on the state $\mathbf{r}$.

Obvious is the following

Lemma 6. *Suppose* $\mathbf{q}s = \mathbf{q}$ *for* m *states* $\mathbf{q}$ *from* Γ *and for some word* s *such that* $s^k = s^{k+1}$*. Then* $|\Gamma s^k| = m$*.*

2 Synchronizing Algorithms

The following help construction was supposed by Eppstein [4]. Let us keep for any pair of states $\mathbf{r}, \mathbf{q}$ the first letter α of the minimal 2-reset word w of the pair of states together with the length of the word w. The corresponding letter of the pair of states $\mathbf{r}\alpha, \mathbf{q}\alpha$ is the second letter of w. The 2-reset word w of minimal length can be restored on this way. The time and space complexity of this preprocessing is $O(|\Gamma^2|)$ [4] and it will be used in majority of considered algorithms.

A help algorithm with $O(|\Gamma|^2 q)$ time complexity in the worst case based on Lemmas 1 and 2 verifies whether or not a given DFA is synchronizing [4], [19]. The main part of the algorithm follows [4] (see also [10]). Our modification of the algorithm finds first all SCC of the graph (a linear algorithm) and then checks the minimal SCC of the graph (if exists). The program for search of automata with relatively great minimal reset word uses this algorithm on the preliminary (and quite often linear) stage.

An efficient semigroup algorithm, essential improvement of the algorithm from [4], based on the properties of syntactic semigroup and inspired by Lemmas 3 - 6 is used on the next stage and plays a central role in the program.

2.1 A Semigroup Algorithm for Synchronizing Word

We consider the square Γ^2 and the reverse graph I of Γ. The graph I is not deterministic for synchronizing graph Γ.

Suppose that the graph Γ is synchronizing, all sink states are found on the stage of checking of the synchronizability, the graph Γ^2 and the reverse graph I were build.

Let us find by help of the reverse graph I for any pair of states $\mathbf{r}, \mathbf{q}$ from Γ^2 the first letter of the minimal 2-reset word w of the pair and the length of w [4]. So for any pair of states $(\mathbf{r}, \mathbf{q})$ can be restored a 2-reset word w of minimal length.

The set of states $(\mathbf{r}, \mathbf{q})$ can be ordered according to the length of the word w. The ordering can be made linear in the size of the set. One can find first the number of all pairs $(\mathbf{r}, \mathbf{q})$ with given length of minimal 2-reset word for any length, then adjust an interval for to place the pairs and then allocate the pairs of states in the interval according to the value of the length.

We use also an another idea for to reorder the pairs of states. The number of preimages of the state $\mathbf{r}w = \mathbf{q}w$ by mapping w^k for any integer k can be used for the ordering together with the length $|w|$. Let us call this order *the second.* The number of preimages can be found in linear time for given pair of states $(\mathbf{r}, \mathbf{q})$ using the reverse graph I. The corresponding words may form a set of generators of a subsemigroup of the semigroup A of all reset words and we will use only linear number of pairs studied for this aim.

The important part of the preprocessing supposed by Eppstein was the computing of the mapping Γw of the graph Γ induced by the minimal 2-reset word w of the pair of states $\mathbf{r}, \mathbf{q}$. This stage begins from the shortest words w and therefore is linear for any considered pair of states $\mathbf{r}, \mathbf{q}$. Nevertheless, the time complexity of the stage is $O(\Gamma^3)$. For to avoid the extremes of this step, our algorithm stops on linear number of pairs. The obtained set G of 2-reset words is considered as a set of generators of some subsemigroup from A and will be marked together with corresponding pairs of states. The time complexity of this step is therefore $O(\Gamma^2)$. Let us reorder G in the *second* order and use the mapping of the graph induced by powers of generators.

Let Γ_i be consecutive images of the graph $\Gamma = \Gamma_0$ such that for $w_i \in A$ holds $\Gamma_i w_{i+1} = \Gamma_{i+1}$ and $|\Gamma_i| > |\Gamma_{i+1}|$. Let A_i be a semigroup generated by the set w_1, ... w_i. Let us check pairs of states corresponding to the words from G. If the pair belongs to Γ_i then the corresponding minimal reset word w_{i+1} may be used for to find the image Γ_{i+1}.

In the case no minimal 2-reset word of a pair from Γ_i was marked, let us consider the products of marked words. If some product unites a pairs of states of Γ_i, then let us use the mapping, mark the product of words and the pair of states. Let us notice that on this step are considered not all marked pairs. The number of considered products must be linear in the size of Γ. The product of two mappings can be found in linear time. Therefore the time complexity of this stage is $O(|\Gamma|k)$ for the defect k of the mapping of Γ_i.

If two considered stages still do not find a reset word, then the new generator must be added to considered subsemigroup A_i. Let us take a pair of states $\mathbf{r}, \mathbf{q}$ from Γ_i with reset word w_i. Suppose $w_i = u_i v_i$ such that the word v_i was marked. Then the mapping w_i can be found in $|\Gamma||u_i|$ time. Let us notice that only on this step the time complexity may by greater than quadratic.

Lemma 7. *Let Γ_i be consecutive images of the graph $\Gamma = \Gamma_0$ such that for v_i from semigroup A $\Gamma_i v_{i+1} = \Gamma_{i+1}$, $|\Gamma_i| > |\Gamma_{i+1}|$ and $|\Gamma_s| = 1$ for some integer s. Let A_i be a semigroup generated by the set w_1, ... w_i such that $w_i = u_i v_i$ is a reset word for some pair of states from Γ_{i-1} and v_i is a marked element of the subsemigroup A_{i-1}.*

Then the considered algorithm has $max(O(|\Gamma|^2 q), O(|\Gamma||u_1 ... u_s|)$ time complexity.

Proof. The time complexity of the step of the building of Γ^2 is $O(|\Gamma|^2 q)$. So $O(|\Gamma|^2 q)$ is a lower bound for the complexity of the considered algorithm.

Let the set w_1, ... w_i generate A_i. The creation of the mapping w_i needs $|\Gamma||u_i| + 1$ steps because for the marked element v_i the mapping is known.

The element will be marked and used only if it is either a generator from A_i or a product of two marked elements. With a marked semigroup element will be associated the mapping of Γ defined by the element. The finding of the mapping of the product of two elements with known images is linear in the size of the graph.

We repeat the process with the obtained image Γ_i. The defect of the mapping is growing on every step. After not over than $|\Gamma| - 1$ steps Γ will be synchronized.

The process of recording of the synchronizing word is linear in the length of the word. The length of the synchronizing word found by the algorithm in billions of practical experiments was less than $|\Gamma|^2$ in all considered cases. The stage of adding of new generators was used only in a small number of cases, only some percents of considered synchronizing automata. The minimal number of generators of the semigroup A is usually small. For instance, for all Černy graphs there are only two generators. Therefore the time complexity of the algorithm is $O(|\Gamma|^2 q)$ in overwhelming majority of cases and the algorithm can be considered as almost quadratic.

2.2 Modification of Eppstein Algorithm

Some version of the program uses also a modification of Eppstein algorithm [4], [10] for finding synchronizing word of $O(|\Gamma|^3 + (|\Gamma|^2 q)$ time complexity. The favorable idea of Eppstein was to keep with any pair of states $\mathbf{r}, \mathbf{q}$ the first letter of the minimal reset word w, its length and the image of the set of states by help of the mapping induced by the word w. The building of the images has $O(|\Gamma|^3)$ time complexity and is a most wasteful part of the algorithm.

Our modification of the Eppstein algorithm (called below a cycle algorithm) instead of a word w considers a power of this word until stabilization of the rank of the image. It proved to be fruitful in many cases including such extraordinary case as graphs of Černy [2]. The length of the reset word obtained by the algorithm in this case reaches its minimum. We omit sometimes this stage of the program despite the growing number of the graphs studied on the next stage. Nevertheless, the observation period of the whole of the program is essentially smaller in spite of the fact that the next stage is non-polynomial.

Theorem 8. *[5], [8] Let C be set of size k and let us consider a sequence of its subsets C_i of size m such that any C_i includes a two-element subset of C not included in every C_j for $j < i$. Then the length of the sequence is less than $(k - m + 2) * (k - m + 1)/2$.*

Corollary 9. *Let Γ be transition graph of an automaton with $|\Gamma|$ states and let us consider a sequence of subsets C_i of states of the automaton of size m or less such that any C_i includes a two-element subset of states of Γ not included in every C_j for $j < i$. Suppose the length of the sequence is $(|\Gamma| - m + 2) * (|\Gamma| - m + 1)/2$. Then at least one C_i contains less than m states. Any sequence of length $(|\Gamma|^3 - |\Gamma|)/6$ of considered kind for distinct m contains a set of size one.*

The value $(|\Gamma|^3 - |\Gamma|)/6$ is well known and was mentioned time and again [5], [8], [9], [12]. The combinatorial theorem 8 can be used for estimation of the length of the reset word obtained by Eppstein, cycle and semigroup algorithms. The theorem considers distinct mappings of the graph of the automaton induced by the letters of the alphabet of the labels such that any new mapping has at least one pair of states that does not belong to any previous mapping of the same rank. For given rank k of mapping in considered algorithms there are at most $(|\Gamma| + k)(|\Gamma| + k - 1)/2$ or less than $|\Gamma|$ distinct mappings. The pair of states with a most short reset word creates a sequence of such mappings and therefore the theorem 8 can be used here. Corollary 9 implies

Proposition 10. *The length of the reset word obtained by Eppstein, cycle and semigroup algorithms is less than* $(|\Gamma|^3 - |\Gamma|)/6$.

So the time complexity of the algorithm in the most worst case is $O(|\Gamma|^3 q)$. Really this most worst case is very rare, for all automata studied for today by these algorithms, it was less than $|\Gamma|^2$.

2.3 An Algorithm for Finding Synchronizing Word of Minimal Length

On the last stage, the program uses a straightforward algorithm for finding synchronizing word of minimal length. The last one is not polynomial in the most worst case (the finding of the synchronizing word of minimal length is NP-hard [4], [10], [15]). The program for search of minimal reset word uses this algorithm relatively rare.

The algorithm is a revision of an algorithm for finding the syntactic semigroup S of size s with q generators on the base of transition graph [17]. We find mappings of the graph of the automaton induced by the letters of the alphabet of the labels. Mappings with the same set of states are identified. It essentially simplified the process in comparison with the algorithm from [17]. Distinct mappings are saved. For this aim, any two mappings must to be compared, so we have $O(s(s-1)/2)$ steps. Let us notice that the size of the syntactic semigroup is in general not polynomial in the size of the transition graph.

The mappings correspond to semigroup elements. With any mapping let us connect a previous mapping and the letter that creates the mapping. On this way, the path on the graph of the automaton can be constructed.

Proposition 11. *The algorithm finds a list of all words (elements of syntactic semigroup) of length k where k is growing. The first synchronizing word of the list has minimal length.*

The time complexity of the considered procedure is $O(|\Gamma| q s^2)$ with $O(|\Gamma| s)$ space complexity.

2.4 Checking Synchronizability

The algorithm is based on the Lemma 1 and presents a modification of an algorithm from [4].

First let us check SCC using the first-depth search and find the SCC Γ_s of sink states from Γ. If there are no sink state then the graph has no synchronizing word and the algorithm stops. Exactly one sink state implies synchronizability and the algorithm also stops. The time and space complexity of these step are linear. Now we can consider the graph Γ_s with at least two sink states.

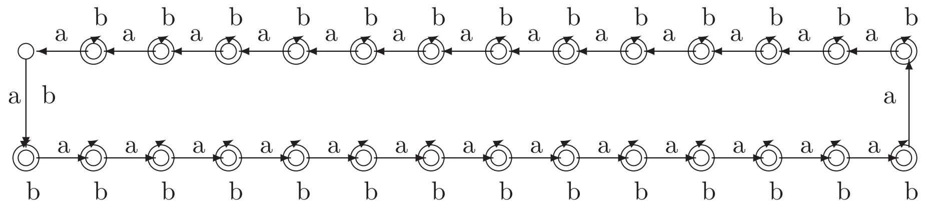
The next step is the consideration of Γ_s^2. We unite any pair of states $(\mathbf{p}, \mathbf{q})$ and $(\mathbf{q}, \mathbf{p})$, all states $(\mathbf{r}, \mathbf{r})$ are united in one state $(0, 0)$. Then let us mark sink state $(0, 0)$ and all ancestors of $(0, 0)$ using the first-depth search on the reverse of the obtained graph G. The graph Γ is synchronizing if any node of G will be marked. The time and space complexity of the algorithm in the most worst case is $O(|\Gamma|^2 q)$.

3 Experimental Data

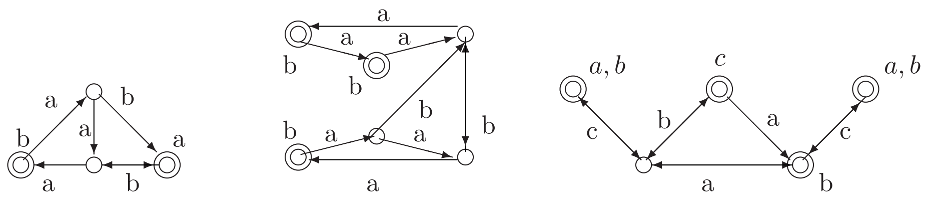
The considered synchronization algorithms were used in a program for search of automata with minimal reset word of relatively great length. The program has investigated all complete DFA for $n \leq 10$, $q = 2$ and for $n \leq 7$, $q \leq 4$.

An automaton with k states outside sink $SCC\ A$ of the transition graph can be mapped on A by word of length not greater than $k(k-1)/2$. Therefore only automata with strongly connected transition graphs need investigation. The graphs with synchronizing proper subgraph obtained by moving off letters from the alphabet are omitted too. The program reduced also the number of studied isomorphic copies of automata. The case of $n = 2$ is not considered because any synchronizing automaton with two states has reset word of length $(n-1)^2 = 1$.

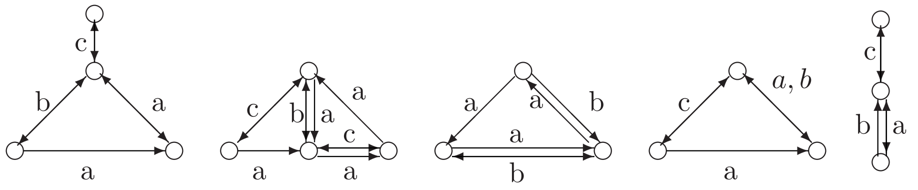
The known n-state automata with minimal reset word of length $(n-1)^2$ are presented by sequence of Černy [2] (here n=28):



by automata supposed by Černy, Piricka and Rosenauerova [3] (CPR), by Kari [6] and Roman [14].



Our program has found five new following examples on the border $(n-1)^2$. The loops of the complete graphs are omitted here for simplicity.



The corresponding reset words of minimal length are: *abcacabca*, *acbaaacba*, *baab*, *acba*, *bacb*. All considered algorithms have found the same reset word for every example. The size of the syntactic semigroup found by the package TESTAS is 148, 180, 24, 27 and 27 correspondingly.

No doubts that some automata from this list, especially for $n = 3$, were sometimes studied by specialists, but we have not found any mention of.

There are no contradictory examples for the Černy conjecture in considered class of automata. Moreover, the program does not find new examples of automata with reset word of length $(n-1)^2$ for $n > 4$ and $q > 3$.

And what is more, the examples with minimal length of reset word disappear even for values near the Černy bound $(n-1)^2$ with growth of the size of the automaton. The gap appears for $n = 6$. There are no 6-state automata with minimal length of synchronizing word equal to 24 for $q \leq 4$.

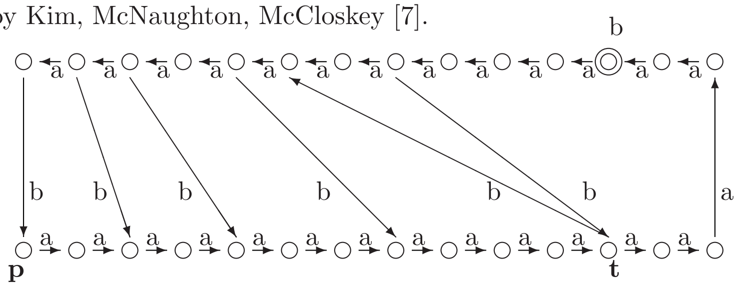
The following table displays this noteworthy trend for the maximum of lengths of minimal reset words. The mentioned above examples on the Černy border are not taken in account in the third line of the table.

size	n=5 $q <= 4$	n=6 $q <= 4$	n=7 $q <= 4$	n=8 q=2	n=9 q=2	n=10 q=2
$(n-1)^2$	16	25	36	49	64	81
max length	15	23	32	44	58	74

The gap between $(n-1)^2$ and the length of the minimal reset word grows with n. This growing gap supports the following funny

Conjecture 1. *The set of n-state DFA* $(n > 2)$ *with minimal reset word of length* $(n-1)^2$ *contains only the sequence of Černy and the eight automata mentioned above, three of size 3, three of size 4, one of size 5 and one of size 6.*

Let us consider the synchronization algorithms from the package TESTAS on some above-mentioned objects and on a modification [16] of a graph KMM supposed by Kim, McNaughton, McCloskey [7].



Complete closure KMML of this graph is obtained from KMM by adding loops in all necessary cases. The n-state automata supposed by Černy will be denoted by C$< n >$.

The following table presents the name of the automaton, the number of its states, the size of the syntactic semigroup, the length of synchronizing word found by the Eppstein algorithm [4], by the cycle and the semigroup algorithm, by the minimal synchronizing word algorithm with the corresponding number of mappings of the set of states.

name	CPR	Roman	Kari	C6	C9	C17	KMM	KMML	C28	C151
graph size	4	5	6	6	9	17	28	28	28	151
semigroup size	145	1397	17265	2742	218718	huge	22126	$> 10^6$	huge	huge
Eppstein alg	9	17	26	27	78	375	4	51	1202	57190
cycle algorithm	9	18	27	25	64	256	4	57	729	22500
semigroup alg	9	17	27	25	64	256	4	27	729	22500
minimal length	9	16	25	25	64	256	4	27	729	22500
mappings	9	22	46	56	501	131053	12	41035	vast	vast

One can compare the results of the algorithms. Equality of the length of minimal synchronizing word and of synchronizing word found by the semigroup algorithm and by Eppstein or cycle algorithm holds in some cases. In particular, it's true even for such extreme objects as Černy automata. Moreover, we obtain not infrequently the same synchronizing words. The transition semigroup of the Černy automaton has a nilpotent element of order $n - 1$, and the minimal synchronizing word of the automaton is a subword of a power of this element.

As for the size of the syntactic semigroup from the table, the most discouraging example gives us the Kari automaton. The size of the syntactic semigroup of the Černy automaton is very great too, it is about $O(2^{2n})$. Maximal size n^n of the syntactic semigroup is reached for the examples of $n = 3$, $q = 3$. It is the semigroup of all transformations of 3-element set.

References

1. D. S. Ananichev, A. Cherubini, M.V. Volkov, An inverse auromata algorithm for recognizing 2-collapsing words. Springer, Lect. Notes in Comp. Sci., 2450(2003),270-282
2. J. Černy, Poznamka k homogenym eksperimentom s konechnymi automatami, Math.-Fyz. Čas., 14(1964) 208-215.
3. J. Černy, A. Piricka, B. Rosenauerova, On directable automata, Kybernetika 7(1971), 289-298.
4. D. Eppstein, Reset sequences for monotonic automata. SIAM J. Comput., **19**(1990) 500-510.
5. P. Frankl, An extremal problem for two families of sets, Eur. J. Comb., 3(1982) 125-127.
6. J. Kari, A counter example to a conjecture concerning synchronizing word in finite automata, EATCS Bulletin, 73(2001) 146-147.

7. Kim S., McNaughton R., McCloskey R. A polynomial time algorithm for the local testability problem of deterministic finite automata, IEEE Trans. Comput., N10, 40(1991) 1087-1093.
8. A.A. Kljachko, I.K. Rystsov, M.A. Spivak, An extremely combinatorial problem connected with the bound on the length of a recurrent word in an automata. Kybernetika. 2(1987) 16-25.
9. Z. Kohavi, J. Winograd, Establishing certain bounds concerning finite automata, J. Comp. System Sci., 7(1973), 288-299.
10. D. Lee, M.Yannakakis, Principle and methods of testing finite state mashines - A survey, Proc. of IEEE, 8, 84(1996) 1090-1123.
11. B.K. Natarajan, An algorithmic approach to the automated design of parts orienters. Proc. of 27th Annual Symp. Foundations of CS, IEEE, 1986, 132-142. Springer, Lect. Notes Comp. Sci., 62(1978) 345-352.
12. J.-E. Pin, On two combinatorial problems arising from automata theory, Annals of Discrete Math., 17(1983) 535-548.
13. J.-K. Rho, F. Somenzi, C. Pixley, Minimum Length Synchronizing Sequences of Finite State Machine, Proc. of 30th ACM/IEEE DA Conf., 1993, 463-466.
14. A. Roman, A note on Cerny Conjecture for automata with 3-letter alphabet (submitted).
15. A. Salomaa, Generation of constants and synchronization of finite automata, J. of Univers. Comput. Sci., 8(2) (2002), 332-347.
16. A.N.Trahtman, Optimal estimation on the order of local testability of finite automata. Theoret. Comput. Sci., 231(2000) 59-74.
17. A.N. Trahtman, Verification of algorithms for checking some kinds of testability. In Algebraic Methods in Language Processing, TWLT 21, eds. F.Spoto, G. Scollo, A. Nijholt. 2003, 253-263.
18. A.N. Trahtman, Černy conjecture for DFA accepting star-free languages. ICALP, Workshop on synchronizing automata, Turku, Finland, 2004.
19. A.N. Trahtman, Some results of implemented algorithms of synchronization. 10-th Journees Montoises d'Inform. Theor., LIege, Belgia, 2004.

On Genome Evolution with Innovation[⋆]

Damian Wójtowicz and Jerzy Tiuryn

Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
{dami, tiuryn}@mimuw.edu.pl

Abstract. We introduce and analyse a simple probabilistic model of genome evolution. It is based on three fundamental evolutionary events: gene duplication, loss and innovation, and it is called *DLI model*. The focus of the paper is around the size distribution of gene families. The formulas for equilibrium gene family sizes are derived showing that they follow a logarithmic distribution. We consider also a disjoint union of DLI models and we present the result of this study. Some empirical results for microbial genomes are presented.

1 Introduction

The seminal Ohno's work [12] shows that gene duplication is a fundamental feature of evolution. It creates the redundancy necessary to free one copy of a gene to evolve a new function, thus it constitutes a substratum for the Darvinian selection for adaptive fitness. This duplication process leads to an appearence of paralogous genes. Recall, that any two genes evolved through a duplication from a single ancestral gene, are called *paralogs*. We do not discuss here the important issue of deciding which genes are paralogous. An in depth discussion of this matter can be found in [4]. Here, we assume that all genes have been already clustered into groups of pairwise paralogous genes. We call such groups *gene or paralog families*. It should be also mentioned that clustering can be made in many different ways [5,14,18,2,1].

The paralog families constitue a significant part of all genes in a genome, about half of the genes have detectable paralogous gene [15]. Therefore the evolution of multigene families has attracted a great deal of attention in comparative genomics during the last decade. It has been noticed that genomes contain gene families of various sizes and these sizes change over time[1]. However, size distribution of gene families in a genome seems to be invariant over time [14,15,5,6,7,8,18]. We propose a model of genome evolution in the spitit of Kimura [9], i.e. in the total absence of selective presures or at least we have to

⋆ This work was partially supported by KBN grants: 3 T11F 021 28 and 3 TF11 016 28.

1 Various biochemical processes (like gene duplication and loss, point mutation, recombination, gene conversion, translocation, horizontal transfer and many others) constantly act on a genome and therefore it changes over time (together with its gene families).

assume that at a genome level, selective pressure does not substantially change the shape of gene family size distribution. We are fully aware that such a purely neutralistic model cannot be truly realistic. However, it explains the biological observations and it can be very useful in further discussions in this topic. The essential feature of our model is that it describes the dynamics of the genome at the level of genes. It is based on three fundamental evolution events:

- *gene duplication* – an event in which one gene gives rise to two genes which cannot be operationally distinguished between themselves; they remain in the same genome and are therefore paralogs;
- *gene loss* – an event which leads to a removal of the gene from the genome;
- *gene innovation* – an event which introduces new genes; it may occur through a horizontal gene transfer of genes between species, especially frequent in case of bacteria or phages [10], or acquired through a series of mutations in a noncoding part of the genome.

We focus on a mathematical analysis of the model from the point of view of paralog family size distribution.

Related Work

A motivation for the present work comes from the study of size distribution of gene familes in several microbial genomes which was undertaken in late 90's. Slonimski *et al.* [14], Huynen and van Nimvegen [5] and Jordan *et al.* [6] counted the number of i-element families of paralogous genes (for $i = 1, 2, 3, \ldots$) in several genomes which have been already sequenced. They came up with different claims concerning the shape of the observed distribution: *logarithmic distribution* in [14,6] (the probability of being an i-element cluster is proportional to θ^i/i, where $0 < \theta < 1$), and *power law distribution* in [5] (the probability is proportional to $i^{-\gamma}$, where $\gamma > 1$). It follows from the above contradicting claims that it may be very difficult to decide what actually is the observed distribution if we rely merely on the biological data. A decisive answer should come by adopting a certain mathematical model of a genome evolution together with a rigorous analysis of the distribution within this model. Yanai *et al.* designed a simple model of the genome evolution, but they show only that it is possible to tune the parameters of the model to obtain the distributions that match closely the observed paralog distributions of the genomes considered by the authors. No mathematical analysis was given in this paper.

To our knowledge the first paper which proposes a model of genome evolution together with complete mathematical analysis of the equilibrium frequences of domain families is the one by Karev *et al.* [7,8]. The model in this paper is similar to our DLI model and is based on three elementary processes: domain birth (duplication), domain death (deletion) and domain innovation, the so called *BDIM model.* Karev *et al.* [7] show in their paper that depending on relative rates of birth and death of domains in families (these rates depend on the size of the family and are constant in time) one obtains various equilibrium distributions, including logarithmic and power law.

In spite of similarities of the BDIM model and the DLI model presented in the present paper, they differ in three important respects: (i) BDIM model is a continuous time process described by a finite system of differential equations, while our model is a discrete time Markov chain with infinitely many states. (ii) BDIM model sets a fixed upper bound on the maximal size of a family, while our model allows families of arbitrary unbounded size. It is not clear what are the consequences for the resulting distribution if one bounds the maximal size of the family. In technical terms bounding the size results in a finite system of differential equations (as this is the case for the BDIM model), while without the bound the system becomes infinite. (iii) Finally, our model presents a particular process of evolution, while BDIM model studies the general quantitative relationships between gene families depending on general relative rates of evolution.

Two models in the spirit of the present paper were analysed in [16] (DL model – a model without the gene innovation event) and [17] (DLC model – a model with the gene accumulated change instead of the gene innovation event). It is shown that the asymptotic distribution in DL model is *geometric* (the probability of being an i-element cluster is proportional to θ^i, where $0 < \theta < 1$), while in DLC model it is a logarithmic distribution. The DL model was treated only as an intermediate step towards an analysis of more complicated models and it shouldn't be surprising that the geometric distribution does not fit the genomic data. Unfortunately, both models, DL and DLC, have an undesirable drawback – depending on the parameters of the model, they have only two types of behaviour: either a colapse of the genome or an exponential explosion (both with probability 1). In order to avoid this unstable behaviour of the genome size in both models, we introduce a gene innovation event. This makes our new model more realistic.

Main Contributions of the Paper
The main result of the paper is the statement that in the presence of gene innovation, the asymptotic size distribution of gene families in the DLI model is the logarithmic distribution whose parameter depends only on the ratio of probability of gene duplication and loss, and does not depend on the rate of innovation. A precise formulation of this result is given in Theorem 1.

The second result of the paper, Theorem 2, shows that a disjoint union of the DLI models result in a linear combination of logarithmic distributions. We also give a forumla for the weights of this combination. This analysis is motivated by the fact that many gene families evolve at different rates and it is natural to introduce different groups of paralog families, each evolving independently according to its own DLI model of evolution with individual parameters.

The paper is organized as follows. In Section 2 we describe the DLI model and present the main result concerning this model. The proof of this result is moved to Section 3. Study of a disjoint union of DLI models is presented in Section 4. Section 5 contains a presentation of the experimental results for five bacteria and five yeasts. Concluding remarks are presented in Section 6.

2 DLI Model of Evolution

In this section we describe formally the model of gene duplication, loss and innovation in a genome, called *DLI model.* We view a genome as a finite collection of genes. We do not assume that genes have any structure – they are treated as atomic objects which undergo various evolutionary events. The events happen randomly and each gene evolves independently of other genes. The whole process is discrete in the sense that we observe in discrete time moments the state of the genome. In order to keep track of evolution of paralog families we assume that each gene has its own color. The intuition behind colors is that two genes have the same color if, and only if, they are paralogs. Hence monochromatic gene families correspond to families of paralogous genes. We assume that at our disposal we have an unlimited supply of colors. A genome is naturally partitioned into gene families according to colors. A *gene family* in a genome is the set of all genes of the genome which have the same color. For $i \geq 1$ let $\mathcal{C}_i$ be the collection of all i-element gene families. In the present paper we are going to study family size distributions. A *family size distribution* of a genome is a probability distribution $(f_i)_{i\geq 1}$ on positive integers, where $f_i = \frac{|\mathcal{C}_i|}{\sum_{k=1}^{\infty}|\mathcal{C}_k|}$ is the probability of observing an i-element family in the genome.

The process of genome evolution consists of two independent subprocesses: internal change and external impact. We describe them separately.

Internal Change: Dupliaction and Loss Process
This is an instance of a Birth and Death process [3]. Fix the real parameters $a, p > 0$ such that $p + ap < 1$. In one step of internal change of a genome each gene of genome is independently:

– *duplicated* with probability ap. Both copies of the gene inherit the color of their parent.
– *lost* with probability p. The gene is removed from the genome.
– remains *unchanged* with probability $1 - p - ap$.

The quantity a is called a *duplication constant.* Remark also that a *lost constant* is assumed to equal 1.

External Impact: Innovation Process
We assume that we have an external source of genes (a black box) which injects genes into the genome randomly according to a certain distribution $(\pi_i)_{i\geq 0}$, whose expected number is $\upsilon = \tau p$ (τ is called an *innovation constant* and p is a probability of gene loss). It means that in one step the innovation process injects i genes with probability π_i. We assume that the injected genes have brand new colors, i.e. colors which do not occur in the genome, and moreover that the colors of injected genes are pairwise different. It will turn out that the choice of the distribution is irrelevant, as long as the expected number $\upsilon = \sum_{i=0}^{\infty} i \cdot \pi_i$ of injected genes is finite and positive. In fact, it will follow from the main result of this paper that the asymptotic gene family size distribution of the innovation process does not even depepend on υ. As we will see later innovation process stabilizes the size of the genome throughout the evolution.

Thus the whole process of genome evolution is described by three positive reals: a, p, and τ, such that $p + ap < 1$. A one step of genome evolution is illustrated in Figure 1.

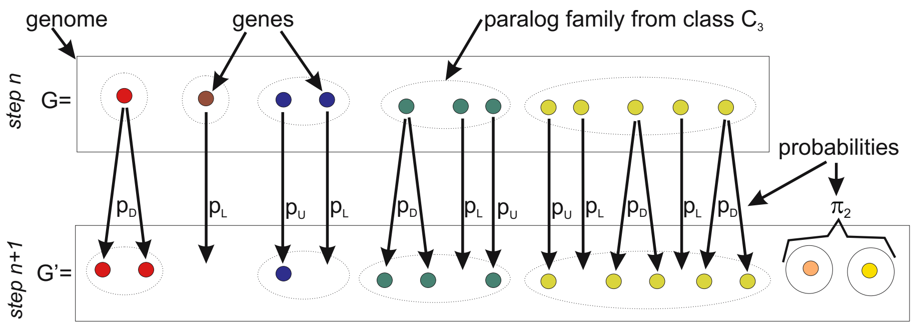


Fig. 1. Illustration of definitions and sample evolution of genome (from G to G'). Here we use the notation $p_L = p$, $p_D = ap$, and $p_U = 1 - p - ap$.

We assume throughout this paper that the initial genome at step $n = 0$ consists of $K > 0$ one element gene families. In order to keep the size of the genome roughly around K during the evolution we need to make some assumptions on the parameters of the model. Observe that when $a > 1$ then the size of the genome grows exponentially fast due to internal change. So the addition of innovation makes things even worse. On the other hand when $a < 1$, then internal change causes the number of genes decrease exponentially fast, but in this case innovation compensates for this loss. This follows from the following simple computation. Given K genes in the genome, in the next step there are on average $(p - ap)K$ genes lost due to internal change and $\upsilon = \tau p$ genes introduced due to innovation. Hence, on average, the size of the genome in the next step is $\tau p + (1 - p + ap)K$. Repeating this argument n times and passing with n to infinity immediately yields the asymptotic size of the genome equal to $\tau/(1 - a)$.

Now we can state the main result of our paper.

Theorem 1. *For any $0 < a < 1$ and $\tau > 0$, if $p > 0$ is sufficiently small, then for a sufficiently large number of evolution steps the observed family size distribution in the genome is close to the logarithmic distribution with parameter a, i.e. the probability of observing a i-element family in the genome is close to $C \cdot a^i/i$, where C is a normalizing constant. Moreover the size of the genome is close to $\tau/(1 - a)$. All the above properties do not depend on the initial size K of the genome, subject to the condition that initially all genes have pairwise different colors.*

The above theorem agrees for $a = 1/2$ with the so called *First Law Of Genomic* stated for small families by Slonimski *et al.* [14,15].

3 Sketch of the Proof of Theorem 1

Let $E_i^{(n)}$ be the expected number of i-element families in the above presented process of evolution after n steps ($i \geq 1, n \geq 0$). Assuming that the initial genome has K genes with pairwise different colors, we have $E_1^{(0)} = K$ and $E_j^{(0)} = 0$ for $j > 1$. For $n > 0$, we obtain an infinite system of equations for $(E_i^{(n)})_{i\geq 1}$:

$$\begin{cases} E_1^{(n)} = \sum_{j=1}^{\infty} E_j^{(n-1)} \cdot \mathbb{P}(j \rightsquigarrow 1) + \tau p \\ E_i^{(n)} = \sum_{j=1}^{\infty} E_j^{(n-1)} \cdot \mathbb{P}(j \rightsquigarrow i) \qquad \text{for } i > 1, \end{cases} \tag{1}$$

where $\mathbb{P}(j \rightsquigarrow i) = \sum_{k=0}^{\lfloor i/2 \rfloor} \binom{j}{i-2k,k} (1-p-ap)^{i-2k}(ap)^k p^{j-i+k}$, for $i, j \geq 1$, is the probability that a gene family of size j gets size i as a result of an internal change of the genome. The situation is slightly different for $i = 1$ since singletons, in addition to the obvious possibility of arriving from other classes by adjusting their size, could have been also created by the gene innovation event.

Let $Q = (\mathbb{P}(j \rightsquigarrow i))_{j,i\geq 1}$ and $T = (\tau p, 0, 0, \ldots)$. System equation (1) can be rewritten in matrix notation: $(E_1^{(n)}, E_2^{(n)}, E_3^{(n)}, \ldots) = (E_1^{(n-1)}, E_2^{(n-1)}, E_3^{(n-1)}, \ldots)Q + T$. It follows that

$$(E_1^{(n)}, E_2^{(n)}, E_3^{(n)}, \ldots) = (K, 0, 0, \ldots)Q^n + T\sum_{i=0}^{n-1} Q^i \tag{2}$$

for all $n \geq 0$. We are interested in the asymptotic distribution which is derived from (2), when n tends to infinity. Notice that *a priori* it is not clear that such a distribution always exists.

For $i \geq 1$, let $q_{p,i}^{(n)}$ be a probability that a random family in the genome after n steps of evolution process has size i. Then

$$q_{p,i}^{(n)} = \frac{E_i^{(n)}}{\sum_{j=1}^{\infty} E_j^{(n)}}. \tag{3}$$

Of course, for every $n \geq 0$ the distribution $\left(q_{p,i}^{(n)}\right)_{i\geq 1}$ exists. The next result says that the asymptotic distribution exists too.

Proposition 1 (Existence of asymptotic distribution). *Let* $0 < a < 1$, $\tau > 0$ *and* $p > 0$. *Then there exists the asymptotic distribution of gene family sizes:*

$$(q_{p,i})_{i\geq 1} = \lim_{n\to\infty} \left(q_{p,i}^{(n)}\right)_{i\geq 1}.$$

Sketch of proof: The idea of this proof is similar to the proof presented in our previous paper [16]. It uses generating functions[2]. Thus, let $f_{p,n}(x)$ be a probabil-

[2] Let $S = (s_i)_{i\in I}$ be a sequence of reals and I be a subset of non-negative integers. A *generating function* for S is a function f defined by power series $f(x) = \sum_{i\in I} s_i x^i$. When S is a probability distribution then the generating function f is called *probability generating function*. See [3].

ity generating function for the distribution $(q_{p,i}^{(n)})_{i\geq 1}$, i.e. $f_{p,n}(x) = \sum_{i=1}^{\infty} q_{p,i}^{(n)} x^i$. We have to show that the limit function $f_p(x) = \lim\limits_{n\to\infty} f_{p,n}(x)$ exists.

We start with the generating function for sequence $E^{(n)} = \left(E_i^{(n)}\right)_{i\geq 1}$. Let $g_{p,n}(x) = K\varphi^{(n)}(x) + \tau p \sum_{i=1}^{n-1} \varphi^{(i)}(x)$, where $\varphi(x) = p + (1-p-ap)x + apx^2$ and $\varphi^{(i)}$ is i-fold composition of φ with itself (with $\varphi^{(0)}$ being the identity function). It follows from equation (2) and Theorem 4 in [16] that the generating function for $E^{(n)}$ is $h_{p,n}(x) = \sum_{i=1}^{\infty} E_i^{(n)} x^i = g_{p,n}(x) - g_{p,n}(0)$. Notice, that $h_{p,n}(1)$ is the number of families in the genome after n steps of the process. Thus we have $f_{p,n}(x) = h_{p,n}(x)/h_{p,n}(1)$.

Next, we prove (using Lemma 1 and Lemma 2 in [16]) that for every $|x| < a^{-1}$, there exists the limit $c_p(x) = \lim_{n\to\infty}(h_{p,n}(1) - h_{p,n}(x))$, which is finite. Moreover, for $x \geq 0$ we have: $c_p(x) = 0$ iff $x = 1$. Thus the limit function $f_p(x)$ exists. □

It can be also proved, using Vitali's Theorem, that f_p is an analytic function with radius of convergence a^{-1}. Moreover, it satisfies the following functional equation

$$f_p(\varphi(x)) = f_p(x) + \frac{\tau p}{c_p(0)}(1-x). \tag{4}$$

See a similar argument in the proof of Theorem 1 in [17]. It should be clear that $c_p(0) = \lim_{n\to\infty} h_{p,n}(1)$ is the asymptotic number of families in the genome.

It follows from the theory of analytic function (the identity property) that function f_p is the unique analytic function which satisfies (4) and the constraint $f_p(0) = 0$ (as well as $f_p(1) = 1$). In this sense, the distribution $\left(q_{p,i}\right)_{i\geq 1}$ is completely characterized by (4). Unfortunately, it cannot be expresed by elementary functions.

Now we can conclude the proof of Theorem 1. It follows from equation (4) that

$$\frac{f_p(\varphi(x)) - f_p(x)}{\varphi(x) - x} = \frac{\tau p}{c_p(0)} \frac{1-x}{\varphi(x) - x} = \frac{\tau}{c_p(0)} \frac{1}{1-ax}.$$

Intuitively it should be clear that the left side of the above equation tends to $f'(x)$, as $p \to 0^+$, because of $\lim_{p\to 0^+} \varphi(x) = x$. Rigorously, it is explained in the proof of Theorem 7 in [16] (see equation (30)). It can be also shown (see a similar proof of Lemma 6 in [17]) that there exists the limit

$$\lim_{p\to 0^+} c_p(0) = \frac{\tau}{C \cdot a}, \tag{5}$$

where $C = (-\ln(1-a))^{-1}$. Thus, we get a differential equation $f'(x) = Ca/(1-ax)$. Solving it, with constraint $f(0) = 0$, we obtain

$$f(x) = C \cdot (-\ln(1-ax)) = C \cdot \sum_{i=1}^{\infty} \frac{a^i}{i} x^i.$$

This completes sketch of the proof of Theorem 1.

4 Disjoint Union of DLI Models

It is well known that gene families evolve at different rates [11,5], and there is a coherent behaviour of genes from one family, i.e. genes from the same family have the same probabilities of duplication and loss. For example, families that are responsible for life processes of an organism possibly do not show propensity to high change over time. Thus it is natural to assume that we have different groups of paralog families, each group evolving according to a DLI model with individual parameters of gene duplication, loss and innovation.

Let $M > 0$ be a number of different groups of gene families in the above sense. Thus we have a family of M DLI models with parameters $(a_m, p_m, \tau_m)_{m=1}^{M}$, where a_m and τ_m are duplication and innovation constants in the m-th group of families, and p_m is a probability of gene loss. We also assume that initially the m-th group has K_m one element gene families, for $m = 1, \dots, M$. Without loss of generality we may assume that $p_m = p$, for $m = 1, \dots, M$. Let us call this model, an *M-DLI model*.

Let $E_{m,i}^{(n)}$ be the expected number of i-element familes in the m-th group of paralog familes after n steps of the M-DLI evolution process. Thus, an expected number of i-element families after n steps of the process equals $E_i^{(n)} = \sum_{m=1}^{M} E_{m,i}^{(n)}$ and the probability of observing in the genome after n steps an i-element family equals (compare it with equation (3)):

$$q_{p,i}^{(n)} = \frac{\sum_{m=1}^{M} E_{m,i}^{(n)}}{\sum_{j=1}^{\infty} \sum_{k=1}^{M} E_{k,j}^{(n)}}. \tag{6}$$

It should be clear that the asymptotic size distribution of paralog families in M-DLC model is a mixture of M logarithmic distributions with parameters a_m $(m = 1, 2, \dots, M)$. We state it precisely in the following theorem.

Theorem 2. *Let $0 < a_m < 1$, $\tau_m > 0$ and $p > 0$, for $m = 1, \dots, M$. Then for a sufficiently small value of p and a sufficiently large number of steps of the evolution process, the size distribution of paralog families in M-DLI model tends to the mixture of logarithmic distributions:*

$$P_{M\text{-}DLI}(i) \approx \sum_{m=1}^{M} \alpha_m \cdot C_m \frac{a_m^i}{i} \qquad i = 1, 2, 3, \dots,$$

where $C_m = (-\ln(1-a_m))^{-1}$ and $\alpha_m = \frac{\tau_m a_m^{-1} C_m^{-1}}{\sum_{k=1}^{M} \tau_k a_k^{-1} C_k^{-1}}$ is an asymptotic fraction of families in the m-th group among all families.

Sketch of proof: We transform equation (6) into $q_{p,i}^{(n)} = \sum_{m=1}^{M} \alpha_{m,p}^{(n)} \cdot q_{m,p,i}^{(n)}$, where $\alpha_{m,p}^{(n)} = \frac{\sum_{j=1}^{\infty} E_{m,j}^{(n)}}{\sum_{k=1}^{M} \sum_{j=1}^{\infty} E_{k,j}^{(n)}}$ and $q_{m,p,i}^{(n)} = \frac{E_{m,i}^{(n)}}{\sum_{j=1}^{\infty} E_{m,j}^{(n)}}$. We have already know, from the analysis of DLI model, that $\lim_{p \to 0^+} \lim_{n \to \infty} q_{m,p,i}^{(n)} = C_m \frac{a_m^i}{i}$. Thus, we have to find $\alpha_m = \lim_{p\to 0^+} \lim_{n\to\infty} \alpha_{m,p}^{(n)}$. It follows from the proof of Proposition 1 that $\alpha_{m,p} = \lim_{n\to\infty} \alpha_{m,p}^{(n)} = \frac{c_{m,p}(0)}{\sum_{k=1}^{M} c_{k,p}(0)}$. Next using (5) we have $\lim_{p\to 0^+} c_{k,p}(0) = \tau_m/(C \cdot a_m)$. This completes sketch of the proof. $\square$

5 Experimental Results

In order to compare the observed families of paralogous genes which occur in species with the values predicted by our model we have examined five bacterial genomes: *Bacillus anthracis Ames*, *Burkholderia mallei ATCC:23344*, *Desulfovibrio vulgaris Hildenborough*, *Geobacter sulfurreducens PCA*, *Pseudomonas putida KT2440*. In addition to this we have also considered five genomes of yeast species: *Candida glabrata*, *Debaromyces hansenii*, *Klyveromyces lactis*, *Saccharomyces cerevisiae* and *Yarrowia lipolytica*, whose genomes (with the exception of *S. cerevisiae*) have been recently sequenced [1]. The bacterial paralogous families were taken from TIGR-CMR [13] web service[3], while the yeast genomes were taken from the CBI web site[4].

As it was observed by many researchers [5,7,14] the distribution of large families of paralogous genes in organisms is very uneven: large families may span hundreds of classes, most of them empty. For this reason some researchers [14,15] restrict an analysis of families to small classes (cluster size 2 through 6), while others [5,7] group families into bins, each containing a certain prespecified minimal number of families. In our analysis we choose the latter method. The observed data was fitted to a mixture of two logarithmic distributions $P_{\log}(i) \propto \beta \frac{a_1^i}{i} + (1-\beta)\frac{a_2^i}{i}$ and the parameters a_1, a_2 and β were choosen to minimize the value of Pearson's χ^2–test. For each genome, before the χ^2–test was evaluated we grouped the expected paralog family frequencies into bins, each containing at least 10 genes. For all analysed genomes, with the exception of *S. cerevisiae*, $P(\chi^2)$ for this model was at least 5%, i.e. no significant difference between the observed and predicted values was detected. The values of parameters a_1, a_2, β and the goodness-of-fit $P(\chi^2)$ for bacterial and yeasts genomes are presented in Tables 1 and 2, respectively.

Tables 1 and 2 can make the impression that distribution parameters for bacterial and yeasts genomes are grouped around different values. Unfortunately this observation reveals an artifact due to the method of clustering paralogous genes. We have experimented (data not shown) with another method of clustering (TribeMCL [2]) which in case of bacteria resulted in a different clustering for which the parameters were similar to those for yeasts. This experiment clearly indicates that the shape of the distribution of paralog families under study may critically depend on the method of clustering. This calls for further investigation, especially in the light of lack of ”golden standards” in this area.

[3] `http://www.tigr.org/tigr-scripts/CMR2/paralog_info_form.spl`

[4] `http://cbi.labri.fr/Genolevures/raw/fam/family-20040327-byfamily.txt`

Table 1. Paralogous families in bacterial genomes [13] and the parameters of best-fit 2-DLI model

Genome	max. fam.	2-DLI model			
	size	a_1	a_2	β	$P(\chi^2)$
Bacillus anthracis Ames	107	0.62	0.95	0.95	80 %
Burkholderia mallei	109	0.67	0.97	0.92	21%
Desulfovibrio vulgaris H.	89	0.63	0.98	0.98	50%
Geobacter sulfurreduncens	108	0.66	0.97	0.96	5%
Pseudomonas putida	110	0.66	0.95	0.89	21%

Table 2. Paralogous families in yeasts genomes [1] and the parameters of best-fit 2-DLI model

Genome	max. fam.	2-DLI model			
	size	a_1	a_2	β	$P(\chi^2)$
Candida glabrata	58	0.35	0.90	0.98	77%
Debaryomyces hansenii	54	0.49	0.94	0.97	5%
Kluyveromyces lactis	49	0.42	0.91	0.97	10%
Saccharomyces cerevisiae	57	0.43	0.93	0.98	2%
Yarrowia lipolytica	48	0.36	0.92	0.97	15%

6 Conclusions

It is really difficult to reach any biological conclusion concerning the shape of size distribution of paralog families by merely fitting the data. Therefore we propose a simple, but a very natural model of genome evolution, which includes three types of events: gene duplication, loss and innovation. We show that the observed family size distribution in the DLI model is close to the logarithmic distribution. What is perhaps little unexpected is that this distribution does not depend on the rate of innovation and only depends on relative rates of duplication *vs.* loss. We present also a disjoint union of DLI models and we conclude that the resulting distribution is a linear combination of the logarithmic distributions.

It is not the intension of our paper to suggest that only a mixture of logarithmic distributions can properly explain the observed data. Genome evolution is a complicated stochastic process which involves many events in addition to the ones considered in this paper. Thus, we do not claim that our model is the most accurate description of this process. Nevertheless it explains well the observed properties of the real data. It would be interesting to see how other evolutionary events, when introduced into the model, affect the asymptotic distribution.

The model without the gene innvoation event but with an accumulated change, the DLC model, has already been analysed in [17]. The model presented here is more realistic. By introducing an innovation, we avoid here the undesirable property of a shrinking or an infinite growth of the genome.

An interesting result of the presented paper is also the fact that knowing only the size of a genome, one can predict the number of gene families in this genome.

References

1. B. Dujon *et al.*, Genome evolution in yeasts. *Nature* 430, pp. 35-44, 2004.
2. A.J. Enright, S. Van Dongen, C.A. Ouzounis, An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research* 30(7), pp. 1575-84, 2002.
3. W. Feller, An introduction to probability theory and its applications. John Wiley and Sons, Inc. New York, London, 1961.
4. W.M. Fitch, Homology, a personal view on some of the problems. *Trends in Genetics*, 16(5), pp. 227-321, 2000.
5. M.A. Huynen, E. van Nimwegen, The Frequency Distribution of Gene Family Size in Complete Genomes. *Molecular Biology Evolution* 15(5), pp. 583–589, 1998.
6. K. Jordan, K.S. Makarova, J.L. Spouge, Y.I. Wolf, E.V. Koonin, Lineage-Specific Gene Expansions in Bacterial and Archeal Genomes. *Genome Research* 11, pp. 555–565, 2001.
7. G.P. Karev, Y.I. Wolf, A.Y. Rzhetsky, F.S. Berezovskaya, E.V. Koonin, Birth and death of protein domains: A simple model of evolution explains power law behavior., *BMC Evolutionary Biology* 2:18, 2002.
8. G.P. Karev, Y.I. Wolf, E.V. Koonin, Simple stochastic birth and death models of genome evolution: was there enough time for us to evolve? *Bioinformatics* 19:15, pp. 1889–1900, 2003.
9. M. Kimura, The Neutral Theory of Molecular Evolution. Cambridge University Press, Cambridge, 1983.
10. Wen-Hsiung Li, Moclecular Evolution. Sinauer Associates, Inc., Publishers, Sunderland Massachusetts, 1997.
11. H. Luz, M. Vingron, Family specific rates of protein evolution. *Bioinformatics* 22(10), pp. 1166-1171, 2006.
12. S. Ohno, Evolution by Gene Duplication. Springer Verlag, Berlin, 1970.
13. J.D. Peterson, L.A. Umayam, T.M. Dickinson, E.K. Hickey, O. White The Comprehensive Microbial Resource. *Nucleic Acids Research* 29:1, pp. 123-125, 2001.
14. P.P. Slonimski, M.O. Mosse, P. Golik, A. Henaût, Y. Diaz, J.L. Risler, J.P. Comet, J.C. Aude, A. Wozniak, E. Glemet, J.J. Codani, The first laws of genomics. *Microbial and Comparative Genomics* 3:46, 1998.
15. P.P. Slonimski, Comparision of complete genomes: Organization and evolution. *Proceedings of the Third Annual Conference on Computational Molecular Biology*, RECOMB'99 Stanislaw Ulam Memorial Lecture, ACM Press, 310, 1999.
16. J. Tiuryn, R. Rudnicki, D. Wójtowicz, A case study of genome evolution: from continuous to discrete time model. In Fiala,J., Koubek,V. and Kratochvíl,J. (eds), *Proceedings of Mathematical Foundations of Computer Science 2004*, LNCS 3153, Springer, pp. 1–24, 2004.
17. J. Tiuryn, D. Wójtowicz, R. Rudnicki, A Model of Evolution of Small Paralog Families in Genomes. (submited for publication), 2006.
18. I. Yanai, C.J. Camacho, C. DeLisi, Predictions of Gene Family Distributions in Microbial Genomes: Evolution by Gene Duplication and Modification. *Physical Review Letters* 85(12), pp. 2641–2644, 2000.

Author Index