Antonín Kučera Thomas A. Henzinger
Jaroslav Nešetřil Tomáš Vojnar
David Antoš (Eds.)

LNCS 7721

Mathematical and Engineering Methods in Computer Science

8th International Doctoral Workshop, MEMICS 2012
Znojmo, Czech Republic, October 2012
Revised Selected Papers

Springer

Antonín Kučera Thomas A. Henzinger
Jaroslav Nešetřil Tomáš Vojnar
David Antoš (Eds.)

Mathematical and Engineering Methods in Computer Science

8th International Doctoral Workshop, MEMICS 2012
Znojmo, Czech Republic, October 25-28, 2012
Revised Selected Papers

Springer

Volume Editors

Antonín Kučera
Masaryk University, Faculty of Informatics
Botanická 68a, 602 00 Brno, Czech Republic
E-mail: tony@fi.muni.cz

Thomas A. Henzinger
Institute of Science and Technology Austria
Am Campus 1, 3400 Klosterneuburg, Austria
E-mail: tah@ist.ac.at

Jaroslav Nešetřil
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
E-mail: nesetril@kam.mff.cuni.cz

Tomáš Vojnar
Brno University of Technology, Faculty of Information Technology
Božetěchova 2, 612 66 Brno, Czech Republic
E-mail: vojnar@fit.vutbr.cz

David Antoš
Masaryk University, Institute of Computer Science
Botanická 68a, 602 00 Brno, Czech Republic
E-mail: antos@ics.muni.cz

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-36044-2 e-ISBN 978-3-642-36046-6
DOI 10.1007/978-3-642-36046-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013930169

CR Subject Classification (1998): D.2.4, C.2.0, F.2.2, G.2.2, C.2.4, K.6.3, K.6.5, K.4.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the proceedings of the 8th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2012) held in Znojmo, Czech Republic, during October 25–28, 2012.

The aim of the MEMICS workshop series is to provide an opportunity for PhD students to present and discuss their work in an international environment. The scope of MEMICS is broad and covers many fields of computer science and engineering. In 2012, submissions were invited especially in the following (though not exclusive) areas:

– Computer-aided analysis and verification
– Applications of game theory in computer science
– Networks and security
– Modern trends of graph theory in computer science
– Electronic systems design and testing
– Quantum information processing

There were 31 submissions from 9 countries. Each submission was thoroughly evaluated by at least three Program Committee members who also provided extensive feedback to the authors. Out of these submissions, 13 papers were selected for publication in these proceedings.

In addition to regular papers, MEMICS workshops also invite PhD students to submit a *presentation* of their recent research results that have already undergone a rigorous peer-review process and have been presented at a high-quality international conference or published in a recognized journal. A total of 28 presentations out of 32 submissions from 11 countries were included in the MEMICS 2012 program.

The MEMICS 2012 program was further enriched by six keynote lectures. The speakers were

– Dirk Beyer from University of Passau with a talk on "CPAchecker: The Configurable Software-Verification Platform"
– Dieter Gollmann from Technische Universität Hamburg-Harburg with a talk on "Security for Cyber-Physical Systems"
– Said Hamdioui from Delft University of Technology with a talk on "Testing Embedded Memories in the Nano-Era: From Defects to Built-In-Self Test"
– Colin McDiarmid from Corpus Christi College Oxford with a talk on "Quicksort and Large Deviations"
– Peter Bro Miltersen from Aarhus University with a talk on "Recent Result on Howard's Algorithm"
– Simon Perdrix from Laboratoire d'Informatique de Grenoble (LIG), CNRS and Université de Grenoble with a talk on "Graph-Based Quantum Secret Sharing"

The MEMICS tradition of *best paper awards* continued also in 2012. The best regular papers were selected at the end of the workshop by the MEMICS 2012 Best Paper Award Committee consisting of Jozef Gruska, Dušan Kolář, Mojmír Křetínský, and Tomáš Vojnar. The winners were:

– Michal Mikuš, STU Bratislava, Slovakia, for the paper “Ciphertext-Only Attack on Gentry-Halevi Implementation of Somewhat Homomorphic Scheme”
– Petr Novotný, MU Brno, Czech Republic, for the paper “Determinacy in Games with Unbounded Payoff Functions”

The awards consisted of a diploma accompanied by a financial prize of approximately 400 Euro. The money were donated by *Red Hat Czech Republic* and *Y Soft*, the MEMICS 2012 Industrial Sponsors.

The MEMICS 2012 workshop was financially supported by the doctoral grant project 102/09/H042 *Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems* from the Czech Science Foundation.

We thank the Program Committee members and the external reviewers for their careful and constructive work. We thank Organizing Committee members who helped to create a unique and relaxed atmosphere which distinguishes MEMICS from other computer science meetings. We also gratefully acknowledge the support of the EasyChair system and the fine cooperation with the *Lecture Notes in Computer Science* team of Springer.

November 2012

Antonín Kučera
Thomas A. Henzinger
Jaroslav Nešetřil
Tomáš Vojnar

Organization

Workshop Organization

The 8th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2012) took place in Znojmo, Czech Republic, on the premises of the Loucký Monastery during October 25–28, 2012. The workshop was attended by 115 participants from 12 countries. More information about the MEMICS workshop series is available at `http://www.memics.cz`.

General Chair

Antonín Kučera	Masaryk University, Brno, Czech Republic

Program Committee Chairs

Thomas A. Henzinger	Institute of Science and Technology, Austria
Jaroslav Nešetřil	Charles University in Prague, Czech Republic
Tomáš Vojnar	Brno University of Technology, Brno, Czech Republic

Program Committee

Andris Ambainis	University of Latvia
Jiří Barnat	Masaryk University, Brno, Czech Republic
Jan Bouda	Masaryk University, Brno, Czech Republic
Patricia Bouyer-Decitre	CNRS, France
Sergio Cabello	University of Ljubljana, Slovenia
Krishnendu Chatterjee	Institute of Science and Technology Austria
Zdeněk Dvořák	Charles University, Prague, Czech Republic
Javier Esparza	TU München, Germany
Rusins Freivalds	University of Latvia
Görschwin Fey	Universität Bremen, Germany
Dieter Gollmann	TU Hamburg-Harburg, Germany
Erich Grädel	RWTH Aachen, Germany
Gregory Z. Gutin	Royal Holloway, University of London, UK
Peter Habermehl	LIAFA, University Paris Diderot, France
Said Hamdioui	TU Delft, The Netherlands
Petr Hanáček	Brno University of Technology, Czech Republic
Holger Hermanns	Saarland University, Germany
Petr Hliněný	Masaryk University, Brno, Czech Republic

Zbyněk Mayer	Masaryk University, Brno, Czech Republic
Ada Nazarejová	Masaryk University, Brno, Czech Republic
Adam Rambousek	Masaryk University, Brno, Czech Republic
Šimon Suchomel	Masaryk University, Brno, Czech Republic

Additional Reviewers

Miklos Bona
Bastian Braun
Mafalda Cortez
Robert Crowston
Mehdi Dehbashi
Vladimír Drábek
Zbyněk Falt
Stefan Frehse
Stephan Huber
Jiří Jaroš
Mark Jones
Syab Khan
Jan Křetínský
Peng Liu
Ashley Montanaro
Rameez Naqvi
Arash Rafiey
Heinz Riener
Sadia Sharmin

Table of Contents

BDD-Based Software Model Checking with CPAchecker

Dirk Beyer and Andreas Stahlbauer

University of Passau, Germany

Abstract. In symbolic software model checking, most approaches use predicates as symbolic representation of the state space, and SMT solvers for computations on the state space; BDDs are sometimes used as auxiliary data structure. The representation of software state spaces by BDDs was not yet thoroughly investigated, although BDDs are successful in hardware verification. The reason for this is that BDDs do not efficiently support all operations that are needed in software verification. In this work, we evaluate the use of a pure BDD representation of integer variable values, and focus on a particular class of programs: event-condition-action systems with limited operations. A symbolic representation using BDDs seems appropriate for this particular class of programs. We implement a program analysis based on BDDs and experimentally compare three symbolic techniques to verify reachability properties of ECA programs. The results show that BDDs are efficient, which yields the insight that BDDs could be used selectively for some variables (to be determined by a pre-analysis), even in general software model checking.

1 Introduction

The internal representation of sets of reachable abstract states is an important factor for the effectiveness and efficiency of software model checking. Binary decision diagrams (BDD) [10] are an efficient data structure for manipulation of large sets, because they represent the sets in a compressed representation, and operations are performed directly on the compressed representation. BDDs are used, for example, to store the state sets in tools for hardware verification [11,12], for transition systems in general [16], for real-time systems [8,13], and push-down systems [14]. There are programming systems for relational programming [2] based on BDDs, and the data structure is used for points-to program analyses [1]. The current state-of-the-art approaches to software verification [3] are either based on satisfiability (SAT) and SAT-modulo-theories (SMT) solving, or on abstract domains from data-flow analysis. BDDs were so far not used as main representation for the state space of integer variables (only as auxiliary data structure). For example, software verifiers based on predicate analysis [4,6] use BDDs for storing truth values of predicates. There exists a version of Java PathFinder that supports the annotation of boolean variables in the program such that the analyzer can track the specified boolean variables using BDDs, which was shown to be efficient for the verification of software product lines [19].

This paper applies BDDs as representation of state sets in the verification of C programs, with a focus on event-condition-action (ECA) systems that use a very limited set of operations. Such ECA programs were used as benchmarks in a recent verification challenge [15][1]. For such a special sub-class of ECA programs, BDDs seem to be promising as representation for two reasons. First, the programs that we consider consist of a single loop in which many conditional branches occur. In each of those branches, a condition is a boolean combination of equalities and negated equalities between variables and values, and an action is a sequence of assignments of values to variables. This means that all required operations are in fact efficiently supported by BDDs, and a symbolic representation using BDDs seems indeed appropriate for this particular class of programs. Second, due to the complex control and data flow of these programs, they are challenging verification tasks for traditional techniques. The formulas that are used as representation in predicate-based approaches represent many paths with a complicated control structure, which are sometimes overwhelming for the SMT solver.

Contribution. We implement a configurable program analysis (CPA) based on BDDs and experimentally compare three symbolic techniques to verify reachability properties of ECA programs. The contribution of this work is not to use BDDs for software verification (which was done before, e.g., in Moped [14]), but to experimentally show that using BDDs as representation for certain variables (which are used in a restricted way) can be more efficient than other (more expressive, but also more expensive) encodings. The insight is that it could be a promising approach to software verification to analyze the usage of each variable in a pre-analysis and then determine for each variable the most efficient representation based on the result.

2 Preliminaries

In order to define a verifier, we need an iteration algorithm and a configurable program analysis, which defines the abstract domain, the transfer relation, as well as the merge and stop operators. In the following, we provide the definitions of the used concepts and notions from previous work [5].

Programs. We consider only a simple imperative programming language, in which all operations are either assignments or assume operations, and all variables are of type integer.[2] We represent a *program* by a *control-flow automaton* (CFA), which consists of a set L of program locations (models the program counter pc), an initial program location pc_0 (models the program entry), and a set $G \subseteq L \times Ops \times L$ of control-flow edges (models the operation that is executed when control flows from one program location to another). The set X of program

[1] `http://leo.cs.tu-dortmund.de:8100/isola2012/`

[2] The framework CPAchecker [6], which we use to implement the analysis, accepts C programs and transforms them into a side-effect free form [18]; it also supports interprocedural program analysis.

variables contains all variables that occur in operations from *Ops*. A *concrete state* of a program is a variable assignment $c : X \cup \{pc\} \to \mathbb{Z}$ that assigns to each variable an integer value. The set of all concrete states of a program is denoted by C. A set $r \subseteq C$ of concrete states is called a *region*. Each edge $g \in G$ defines a (labeled) transition relation $\xrightarrow{g} \subseteq C \times \{g\} \times C$. The complete transition relation $\to$ is the union over all control-flow edges: $\to = \bigcup_{g \in G} \xrightarrow{g}$. We write $c \xrightarrow{g} c'$ if $(c, g, c') \in \to$, and $c \to c'$ if there exists a g with $c \xrightarrow{g} c'$. A concrete state c_n is *reachable* from a region r, denoted by $c_n \in Reach(r)$, if there exists a sequence of concrete states $\langle c_0, c_1, \ldots, c_n \rangle$ such that $c_0 \in r$ and for all $1 \leq i \leq n$, we have $c_{i-1} \to c_i$. Such a sequence is called *feasible program path*. In order to define an efficient program analysis, we need to define abstract states and abstract transitions.

Configurable Program Analysis. We use the framework of *configurable program analysis* (CPA) [5] to formalize our program analysis. A CPA specifies the abstract domain and a set of operations that control the program analysis. A CPA is defined independently of the analysis algorithm, and can be plugged in as a component into the software-verification framework without working on program parsers, exploration algorithms, and other general data structures. A CPA $\mathbb{C} = (D, \rightsquigarrow, \mathsf{merge}, \mathsf{stop})$ consists of an abstract domain D, a transfer relation $\rightsquigarrow$ (which specifies how to compute abstract successor states), a merge operator merge (which defines how to merge abstract states when control flow meets), and a stop operator stop (which indicates if an abstract state is covered by another abstract state). The abstract domain $D = (C, \mathcal{E}, [\![\cdot]\!])$ consists of a set C of concrete states, a semi-lattice $\mathcal{E}$ over abstract-domain elements, and a concretization function that maps each abstract-domain element to the represented set of concrete states. The abstract-domain elements are also called *abstract states*.

Using this framework, program analyses can be composed of several component CPAs. We will now give the definition of a location analysis; our complete analysis will be the composition of the location analysis with the BDD-based analysis that we will define later.

CPA for Location Analysis. The CPA for *location analysis* $\mathbb{L} = (D_{\mathbb{L}}, \rightsquigarrow_{\mathbb{L}}, \mathsf{merge}_{\mathbb{L}}, \mathsf{stop}_{\mathbb{L}})$ tracks the program counter pc explicitly [5].

1. The domain $D_{\mathbb{L}}$ is based on the flat semi-lattice for the set L of program locations: $D_{\mathbb{L}} = (C, \mathcal{E}_{\mathbb{L}}, [\![\cdot]\!])$, with $\mathcal{E}_{\mathbb{L}} = ((L \cup \{\top\}), \sqsubseteq)$, $l \sqsubseteq l'$ if $l = l'$ or $l' = \top$, $[\![\top]\!] = C$, and for all l in L, $[\![l]\!] = \{c \in C \mid c(pc) = l\}$.

2. The transfer relation $\rightsquigarrow_{\mathbb{L}}$ has the transfer $l \overset{g}{\rightsquigarrow}_{\mathbb{L}} l'$ if $g = (l, \cdot, l')$.

3. The merge operator does not combine elements when control flow meets: $\mathsf{merge}_{\mathbb{L}}(l, l') = l'$.

4. The termination check returns true if the current element is already in the reached set: $\mathsf{stop}_{\mathbb{L}}(l, R) = (l \in R)$.

Analysis Algorithm. Algorithm 1 shows the core iteration algorithm that is used to run a configurable program analysis, as implemented by tools like CPAchecker. The algorithm is started with a CPA and two sets of abstract

Algorithm 1. $CPA(\mathbb{D}, R_0, W_0)$ (taken from [5])

Input: a CPA $\mathbb{D} = (D, \rightsquigarrow, \mathsf{merge}, \mathsf{stop})$,
a set $R_0 \subseteq E$ of abstract states,
a subset $W_0 \subseteq R_0$ of frontier abstract states,
where E denotes the set of elements of the semi-lattice of D

Output: a set of reachable abstract states,
a subset of frontier abstract states

Variables: two sets reached and waitlist of elements of E

```
1:  reached := $R_0$;
2:  waitlist := $W_0$;
3:  while waitlist $\neq \emptyset$ do
4:    choose $e$ from waitlist; remove $e$ from waitlist;
5:    for each $e'$ with $e \rightsquigarrow e'$ do
6:      for each $e'' \in$ reached do
7:        // Combine with existing abstract state.
8:        $e_{new}$ := merge($e', e''$);
9:        if $e_{new} \neq e''$ then
10:         waitlist := (waitlist $\cup \{e_{new}\}) \setminus \{e''\}$;
11:         reached := (reached $\cup \{e_{new}\}) \setminus \{e''\}$;
12:     // Add new abstract state?
13:     if $\neg$ stop($e'$, reached) then
14:       waitlist := waitlist $\cup \{e'\}$;
15:       reached := reached $\cup \{e'\}$;
16: return (reached, waitlist)
```

states as input: the set R_0 (reached) contains the so far reached abstract states, and the set W_0 (waitlist) contains abstract states that the algorithm needs to process. The algorithm terminates if the set waitlist is empty (i.e., all abstract states are processed) and returns the two sets reached and waitlist. We start the algorithm with two singleton sets that contain only the initial abstract state. In each iteration of the 'while' loop, the algorithm processes and removes one state e from the waitlist, by computing all abstract successors and further processing them as e'.

Next, the algorithm checks (lines 6–11) if there is an existing abstract state in the set of reached states with which the new state e' has to be merged (e.g., where control flow meets after completed branching). If this is the case, then the new, merged abstract state is substituted for the existing abstract state in both sets reached and waitlist. (This operation is sound because the merge operation is not allowed to under-approximate.) In lines 12–15, the stop operator checks if the new abstract state is covered by a state that is already in the set reached, and inserts the new abstract state into the work sets only if it is not covered.

Binary Decision Diagrams. A binary decision diagram (BDD) [10] represents a set of assignments for a set of boolean variables. In our analysis, we need to consider integer variables. We encode the integer assignments as bit vectors, and the integer variables as vectors of boolean variables, and thus, can represent data states of integer programs by BDDs.

A BDD is a rooted directed acyclic graph, which consists of decision nodes and two terminal nodes (called 0-terminal and 1-terminal). Each decision node is labeled by a boolean variable and has two children (called low child and high child). A BDD is maximally reduced according to the following two rules: (1) merge any isomorphic sub-graphs, and (2) eliminate any node whose two children are isomorphic. Every variable assignment that is represented by a BDD corresponds to a path from the root node to the 1-terminal. The variable of a node has the value 0 if the path follows the edge to the low child, and the value 1 if it follows the edge to the high child. A BDD is always ordered, which means that the variables occur in the same order on any path from the root to a terminal node. For a given variable order, the BDD representation of a set of variable assignments is unique.

3 BDD-Based Program Analysis

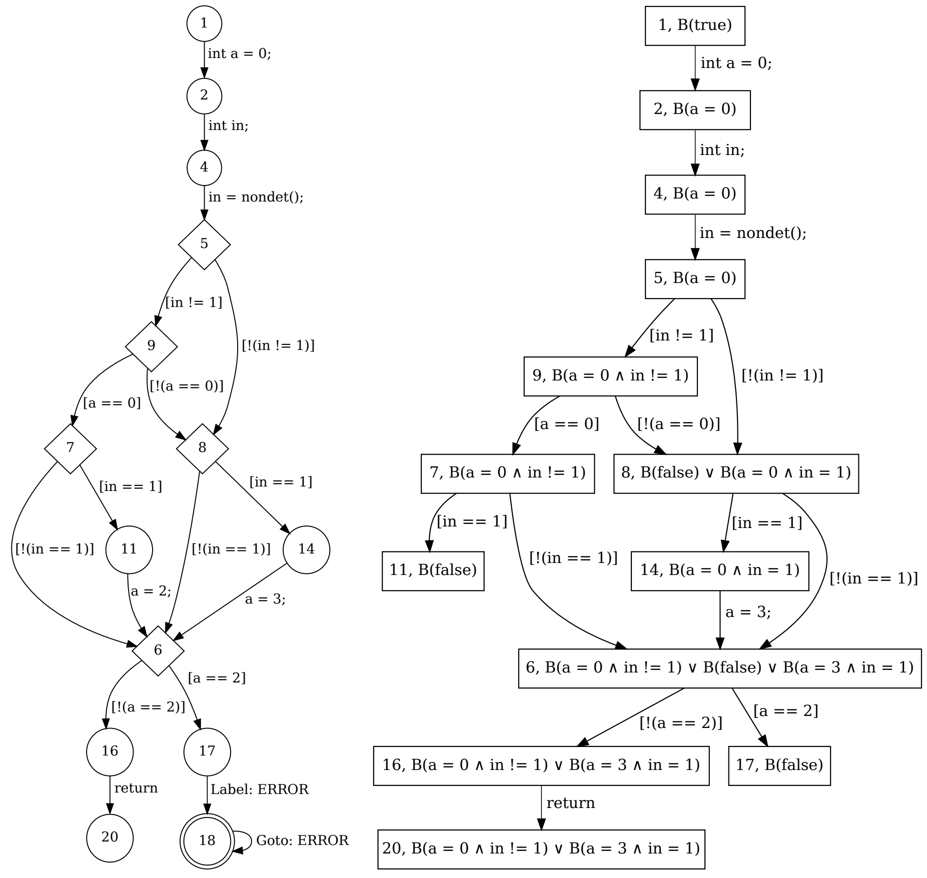
For implementing the BDD-based analysis, we define a configurable program analysis (CPA) that uses BDDs to represent abstract states, and implement it in the open-source tool CPAchecker.

Let X be the set of program variables. Given a first-order formula φ over X, we use $\mathbb{B}_\varphi$ to denote the BDD that is constructed from φ, and $[\![\varphi]\!]$ to denote all variable assignments that fulfill φ. Given a BDD $\mathbb{B}$ over X, we use $[\![\mathbb{B}]\!]$ to denote all variable assignments that $\mathbb{B}$ represents ($[\![\mathbb{B}_\varphi]\!] = [\![\varphi]\!]$).

The *BDD-based program analysis* is a configurable program analysis $\mathbb{BPA} = (D_{\mathbb{BPA}}, \rightsquigarrow_{\mathbb{BPA}}, \mathsf{merge}_{\mathbb{BPA}}, \mathsf{stop}_{\mathbb{BPA}})$ that represents the data states of the program symbolically, by storing the values of variables in BDDs. The CPA consists of the following components:

1. The abstract domain $D_{\mathbb{BPA}} = (C, \mathcal{E}, [\![\cdot]\!])$ is based on the semi-lattice $\mathcal{E}_\mathbb{B}$ of BDDs, i.e., every abstract state consists of a BDD. The concretization function $[\![\cdot]\!]$ assigns to an abstract state $\mathbb{B}$ the set $[\![\mathbb{B}]\!]$ of all concrete states that are represented by the BDD. Formally, the lattice $\mathcal{E}_\mathbb{B} = (\mathcal{B}, \sqsubseteq)$ —where $\mathcal{B}$ is the set of all BDDs, $\mathbb{B}_{true}$ is the BDD that represents all concrete states (1-terminal node), and $\mathbb{B}_{false}$ is the BDD that represents no concrete state (0-terminal node)— is induced by the partial order $\sqsubseteq$ that is defined as: $\mathbb{B} \sqsubseteq \mathbb{B}'$ if $[\![\mathbb{B}]\!] \subseteq [\![\mathbb{B}']\!]$. (The join operator $\sqcup$ yields the least upper bound; $\mathbb{B}_{true}$ is the top element $\top$ of the semi-lattice.)
2. The transfer relation $\rightsquigarrow_{\mathbb{BPA}}$ has the transfer $\mathbb{B} \overset{g}{\rightsquigarrow} \mathbb{B}'$ with
$$\mathbb{B}' = \begin{cases} \mathbb{B} \wedge \mathbb{B}_p & \text{if } g = (l, \texttt{assume}(p), l') \\ (\exists w : \mathbb{B}) \wedge \mathbb{B}_{w=e} & \text{if } g = (l, \texttt{w} := \texttt{e}, l') \end{cases}.$$
3. The merge operator is defined by $\mathsf{merge}_{\mathbb{BPA}}(\mathbb{B}, \mathbb{B}') = \mathbb{B} \vee \mathbb{B}'$.
4. The termination check is defined by $\mathsf{stop}_{\mathbb{BPA}}(\mathbb{B}, R) = \exists \mathbb{B}' \in R : \mathbb{B} \sqsubseteq \mathbb{B}'$.

We construct the complete program analysis by composing the CPA $\mathbb{BPA}$ for BDD-based analysis with the CPA $\mathbb{L}$ for location analysis, in order to also track the program locations. For further details on CPA composition, we refer to [5].



(a) Control-flow autom. (CFA) (b) Abstract reachability graph (ARG)

Fig. 1. Example program with verification certificate

Example. Consider the program represented by the control-flow automaton (CFA) in Fig. 1(a). The error location (location 18, indicated by label 'ERROR') is not reachable in this simple example program, i.e., the program is safe. Figure 1(b) represents the corresponding abstract-reachability graph (ARG), which could serve as verification certificate for this analysis. The nodes in the ARG represent abstract states, which are initial abstract states or constructed by computing abstract successor states according to the edges of the CFA, using the CPA algorithm and composition of CPAs as described above. The edges in the ARG represent successor computations along the control-flow edges of the corresponding CFA. We label each node of the ARG with the program location and the BDD that represents the abstract data state. The set of states that are represented by the nodes of the ARG shown in Fig. 1(b) equals the set reached after the CPA algorithm has terminated.

The analysis starts at the initial program location $pc_0 = 1$ with the initial abstract data state e_0, which is represented by the BDD $\mathbb{B}_{true}$. The analysis then computes the abstract successor states by applying the transfer relation $\rightsquigarrow$; in our example the abstract data state for location $pc = 2$ is computed by quantifying the assigned variable in the BDD of the previous abstract state, create a BDD for the constraint of control-flow edge `int a=0` (assignment) and conjunct it with the former BDD. The transition along the edge $(2, \texttt{int in}, 4)$ does not change the abstract data state because the variable that is declared by this edge was not known before; also the transition along $(4, \texttt{in = nondet()}, 4)$ does not change the data state because it does not restrict the possible concrete states (the return value of `nondet()` is non-deterministic). Transitions whose operations are assumptions, for example, $(5, \texttt{[in != 1]}, 9)$ are encoded by conjuncting the BDD $\mathbb{B}$ of the abstract data state of the predecessor location ($pc = 5$) with the BDD for the respective assumption (`in != 1`), i.e., the successor state $\mathbb{B}'$ is computed as $\mathbb{B}' = \mathbb{B}_{a=0} \wedge \mathbb{B}_{in\ != 1}$. Now we consider, for example, location $pc = 14$, which has the BDD $\mathbb{B}_{a=0\ \wedge\ in=1}$ as abstract data state, and process the control-flow edge $(14, \texttt{a = 3}, 6)$ (assignment). Assignment operations are processed by first existential quantifying the variable that gets a new value assigned (`a`); then the intermediate BDD $\mathbb{B}_{in=1}$ is conjuncted with the BDD that represents the new value of the variable: $\mathbb{B}' = \mathbb{B}_{in=1} \wedge \mathbb{B}_{a=3}$.

Abstract states that were computed for the same program location are —as defined by the CPA operator merge— joined by computing the disjunction of the BDDs; the abstract data state $\mathbb{B}_{(a=0\ \wedge\ in\ != 1) \vee (false) \vee (a=3\ \wedge\ in=1)}$ at location $pc = 6$ is such a result of a join. After the analysis has terminated, the set reached of reached states contains at most one abstract state for each program location.

The computation of successors of a given abstract state e stops (the abstract state is not added to the sets waitlist and reached for further processing), whenever the abstract data state is already covered by (implies) an existing abstract data state; this check is performed by the CPA operator stop. The analysis does not process successors of locations 11 and 17, because the BDDs evaluate to *false*. Thus, the error location 18 is not reached.

4 Evaluation

In order to demonstrate that the BDD-based analysis yields a significant performance improvement on a set of C programs with restricted operations on integer variables, we compare our simple analysis with two other approaches for symbolic software model checking.

Experimental Setup. All experiments were performed on machines with a 3.4 GHz Quad Core CPU and 16 GB of RAM. The operating system was Ubuntu 12.04 (64 bit), using Linux 3.2.0-30 and OpenJDK 1.6.0_24. A time limit of 5 min and a memory limit of 15 GB were used. We took CPAchecker from revision 6607 of the trunk in the repository, and MathSAT 4.2.17 as SMT solver; for the BDD-based analysis we configured it with a Java heap size of 13 GB, for the other analyses we configured it with 10 GB, in order to leave RAM for the SMT solver.

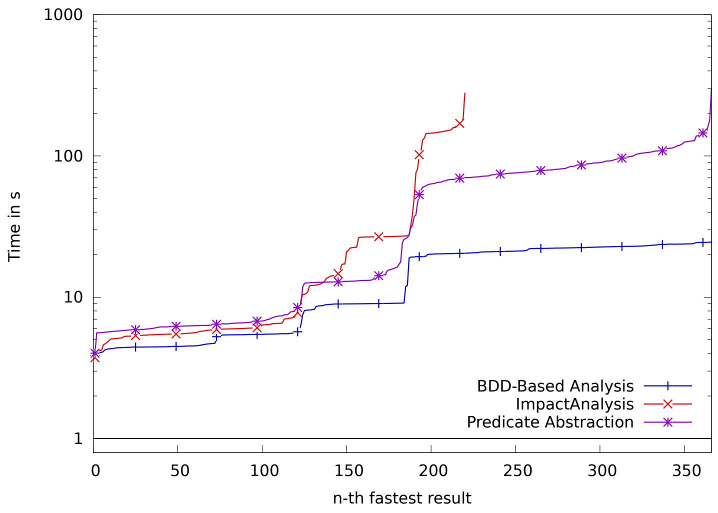


Fig. 2. Quantile functions for the three different approaches

The configuration of the BDD-based analysis is specified in the configuration file `fsmBddAnalysis.properties`.

Verification Tasks. For the evaluation of our approach, we use Problems 1 to 6 from the recent RERS challenge, because those programs are in the restricted class of C programs that we described earlier. Tables with detailed results and the benchmark programs are publicly available on the accompanying web page at `http://www.sosy-lab.org/∼dbeyer/cpa-bdd`.

Compared Verification Approaches. We restrict the comparison to three symbolic techniques that are all implemented in the same verification tool, in order to eliminate influence of the used solver, libraries, parser front-ends, etc. The first approach is an Impact-based analysis [17]. This analysis is based on counterexample-guided abstraction refinement (CEGAR) and computes abstractions using interpolation along infeasible error paths. In contrast to predicate abstraction, this analysis does not compute strongest post-conditions and abstracts those to more abstract formulas, but uses a conjunction of the obtained interpolants as abstract states. A detailed comparison of the approach with predicate abstraction can be found in the literature [9]. The second approach is based on CEGAR and predicate abstraction, together with adjustable-block encoding [7]. The third approach is the BDD-based analysis that was introduced in this paper.

Discussion. Figure 2 gives an overview over the results using a quantile plot of the verification times (all verification tasks, no separation between satisfied and violated properties). A quantile plot orders, for each approach separately, the verification runs by the run time that was needed to obtain the correct verification result on the x-axis (n-th fastest result). A data point (x, y) of the graph means that x verification tasks were successfully verified each in under y

Table 1. Detailed results for the verification tasks with result 'UNSAFE'

		Impact Algorithm			Predicate Abstraction			BDD-Based Analysis		
Problem	# Properties	Solved properties	Time (total)	Time (mean)	Solved properties	Time (total)	Time (mean)	Solved properties	Time (total)	Time (mean)
Problem 1	14	14	86	6.2	14	100	7.4	14	65	4.6
Problem 2	8	8	37	4.7	8	48	6.0	8	33	4.2
Problem 3	14	10	120	12	14	190	13	14	110	8.2
Problem 4	25	1	14	14	25	2600	100	25	490	20
Problem 5	25	25	3600	150	25	3200	130	25	520	21
Problem 6	26	2	100	52	26	2200	85	26	520	20

Table 2. Detailed results for the verification tasks with result 'SAFE'

		Impact Algorithm			Predicate Abstraction			BDD-Based Analysis		
Problem	# Properties	Solved properties	Time (total)	Time (mean)	Solved properties	Time (total)	Time (mean)	Solved properties	Time (total)	Time (mean)
Problem 1	47	47	270	5.8	47	310	6.7	47	260	5.5
Problem 2	53	53	320	6.0	53	320	6.0	53	240	4.5
Problem 3	47	17	230	14	47	660	14	47	420	9.0
Problem 4	36	36	950	27	36	2500	70	36	820	24
Problem 5	36	6	790	130	36	2700	75	36	800	22
Problem 6	35	1	51	51	35	2800	80	35	840	24

seconds of CPU time. The integral below the graph illustrates the accumulated verification time for all solved verification tasks. The Impact-based analysis is not able to solve all verification tasks (it solves 220 instances), the predicate-abstraction-based analysis can verify each property within 300 s of CPU time. The BDD-based analysis, which we introduced earlier in this paper, is able to solve each of the properties within 25 s.

Table 1 shows more detailed results for the violated properties, i.e., the verification tasks for which the verification result is 'UNSAFE', and Table 2 shows the details for the satisfied properties. The verification time (total and mean) values are given in seconds of CPU time with two significant digits. The Impact-based analysis can solve the verification tasks of the programs Problem 1 and Problem 2 completely; performance and precision decrease dramatically for Problems 3 to 6. The predicate-abstraction-based analysis and the BDD-based analysis can both verify all properties; but the BDD-based analysis is significantly more efficient. The BDD-based analysis scales best with the problem size (assuming that the verification tasks for the program 'Problem $n+1$' are harder than the tasks for the program 'Problem n').

5 Conclusion

We extended the standard software-verification tool CPAchecker by a configurable program analysis (CPA) that uses BDDs as data structure to represent sets of data states (variable assignments). We have compared the effectiveness and efficiency of this analysis to other approaches that use symbolic techniques: a program analysis that computes abstract successor states using predicate abstraction after every successor computation [7], and a program analysis that computes abstract states along paths using interpolation [9,17] — both being state-of-the-art approaches for symbolic software verification.

The experiments show that the BDD-based approach is the most efficient verification approach (by an order of magnitude) for the considered class of programs. However, as soon as the programs use more general operations, BDDs would be prohibitively less efficient. This means that BDDs can be more efficient than other representations for certain types of variables (the ones that are involved in simple operations only). This is an important insight and motivation for future work: It would be promising to pre-analyze the program in order to find out for each variable how it is used, and then determine —based on its usage-type— the most efficient abstract domain to track this variable. The other variables can be analyzed by an explicit-value analysis or a predicate-analysis; using a configuration program analysis (CPA) with adjustable precisions; such combinations of program analyses are easy to construct in CPAchecker.

References

1. Berndl, M., Lhoták, O., Qian, F., Hendren, L., Umanee, N.: Points-to Analysis using BDDs. In: Proc. PLDI, pp. 103–114. ACM (2003)
2. Beyer, D.: Relational Programming with CrocoPat. In: Proc. ICSE, pp. 807–810. ACM (2006)
3. Beyer, D.: Competition on Software Verification (SV-COMP). In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 504–524. Springer, Heidelberg (2012)
4. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The Software Model Checker Blast. Int. J. Softw. Tools Technol. Transfer 9(5-6), 505–525 (2007)
5. Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 504–518. Springer, Heidelberg (2007)

6. Beyer, D., Keremoglu, M.E.: CPAchecker: A Tool for Configurable Software Verification. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 184–190. Springer, Heidelberg (2011)
7. Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate Abstraction with Adjustable-Block Encoding. In: Proc. FMCAD, pp. 189–197 (2010)
8. Beyer, D., Lewerentz, C., Noack, A.: Rabbit: A Tool for BDD-Based Verification of Real-Time Systems. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 122–125. Springer, Heidelberg (2003)
9. Beyer, D., Wendler, P.: Algorithms for Software Model Checking: Predicate Abstraction vs. IMPACT. In: Proc. FMCAD (2012)
10. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
11. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L.: Sequential Circuit Verification using Symbolic Model Checking. In: Proc. DAC, pp. 46–51. ACM (1990)
12. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic Model Checking: 10^{20} States and Beyond. In: Proc. LICS, pp. 428–439. IEEE (1990)
13. Campos, S.V.A., Clarke, E.M.: The Verus Language: Representing Time Efficiently with BDDs. In: Rus, T., Bertrán, M. (eds.) ARTS 1997. LNCS, vol. 1231, pp. 64–78. Springer, Heidelberg (1997)
14. Esparza, J., Kiefer, S., Schwoon, S.: Abstraction Refinement with Craig Interpolation and Symbolic Pushdown Systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 489–503. Springer, Heidelberg (2006)
15. Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS Grey-Box Challenge 2012: Analysis of Event-Condition-Action Systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 608–614. Springer, Heidelberg (2012)
16. McMillan, K.L.: The SMV System. Technical Report CMU-CS-92-131, Carnegie Mellon University (1992)
17. McMillan, K.L.: Lazy Abstraction with Interpolants. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)
18. Necula, G.C., McPeak, S., Rahul, S.P., Weimer, W.: CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. In: Horspool, R.N. (ed.) CC 2002. LNCS, vol. 2304, pp. 213–228. Springer, Heidelberg (2002)
19. von Rhein, A., Apel, S., Raimondi, F.: Introducing Binary Decision Diagrams in the Explicit-State Verification of Java Code. In: Proc. Java Pathfinder Workshop (2011)

Security for Cyber-Physical Systems
(Extended Abstract)

Dieter Gollmann

Institute for Security in Distributed Applications
Hamburg University of Technology
Hamburg, Germany
diego@tu-harburg.de

Abstract. Cyber-physical systems are characterized by an IT infrastructure controlling effects in the physical world. Attacks are intentional actions trying to cause undesired effects in the physical world. We examine to which extent traditional IT security techniques can protect against attacks on cyber-physical systems and which additional measures could be deployed to strengthen their security. Properties relevant in this context are the veracity and plausibility of the data being processed.

Keywords: Critical infrastructures, security in cyber-physical systems, veracity, plausibility.

All science is either physics or stamp collecting. [Ernest Rutherford]

Cyber-physical systems, a.k.a. *embedded systems*, are IT systems "embedded" in an application in the physical world. They consist of sensors, actuators, the processors that are part of the control system, and communication networks. Some of these systems are critical because they are part of an infrastructure critical for society. Critical infrastructure protection has been a high profile topic for a decade at least.

There exists, for example, a considerable body of work on SCADA (Supervisory Control and Data Acquisition) systems. Investigations into SCADA security are often motivated by the observation that critical systems which were once isolated installations using proprietary protocols are now being connected to the Internet (which we all know to be terribly insecure). The focus is thus very much on securing the communications infrastructure, and partly also on software in control systems. Most of these papers are surveys by nature, applying known security techniques to known security problems. Such reports are undoubtedly justified as warnings to an industry new to the Internet, but novel research challenges are hardly identified.

What is then missing? The physical world, the sensors and actuators, are not part of the picture. The Stuxnet worm [1] did not reach its target on the Internet but on a physical device (USB stick). Stuxnet came with a digital certificate and was injected by a "trusted" party, i.e. a contractor with legitimate access to the site. The standard armoury in network security, viz. firewalls, secure tunnels, digital signatures, and access control, provided no defence.

Cryptography belongs to cyber space. "Trusting" sensor data just because they have been digitally signed does not help when the sensor readings have been manipulated before reading. "Trusting" code just because it has been digitally signed does not help when the code is vulnerable or outright malicious[1]. Encrypting sensor data does not help much when the adversary can take the same environmental readings with moderate effort[2].

We have to include the physical world in our considerations. In this respect, we should not look at the cyber-part of the cyber-physical system as an infrastructure that needs protection from attacks in cyber space. We would miss the view on what happens in the physical world. The cyber-part of a cyber-physical system is better treated as a control system. This view encourages us to think about effects in the physical world and about ways of tampering with inputs before they are handed to the IT infrastructure.

We have arrived at an old problem in a new setting. How can we be sure that the inputs from sensor readings faithfully capture reality? We will call this property *veracity*. In the past research on reliability had been concerned with accidental sensor failures. Replication (redundancy) and *consistency* checks such as majority voting had been used to detect components reporting wrong data. These defences build on the assumption of the statistical independence of component failures.

With intentional attacks the efficacy of defences cannot be argued on the basis of statistical independence. We may then use models of the physical space to be controlled that relate different types of sensor inputs (different aspects of the physical world) and perform *plausibility* checks that judge to which extent individual sensor readings are consistent with the general view on the system derived from all the readings. A security argument would then have to show how difficult it is to deceive the control system about the state of the system being controlled. Attacks may be launched in the physical world – by manipulating sensors or the environment around a sensor – and in the control system (here we are back to traditional IT security). The laws of physics may help the defender by excluding impossible sensor readings or flagging implausible readings as suspicious. Tamper resistance (hardware and software) and physical security are other familiar principles that can be part of the defence.

In a cyber-physical systems we further have to ensure that actuators are working as instructed. The established approach places sensors in the system to observe the actions of actuators. We are then back to verifying the veracity of sensor readings. In the "old" days where controllers communicated with actuators via analogue signals there was not much more that could be done.

[1] This is old news. It was argued in the late 1990s that digitally signed code, e.g. Microsoft's Authenticode, does not provide relevant security guarantees whereas sandboxing could do so.

[2] Much work on WSN security fails to justify why the attacker has no viable alternatives to intercepting WSN traffic to obtain information from the physical world.

Today, with layered software architectures we may check whether instructions issued at a higher layer are faithfully translated into a lower layer language understood by smart instruments.

Measures to secure critical cyber-physical systems have to go beyond securing the IT infrastructure and check the veracity of the inputs used for controlling the system. We have sketched some generic principles but concrete solutions are likely to be specific, building on physical aspects of the system being controlled.

Reference

1. Langner, R.: Stuxnet: Dissecting a cyberwarfare weapon. IEEE Security & Privacy 3(9), 49–51 (2011)

Quantum Secret Sharing with Graph States

Sylvain Gravier[1,2], Jérôme Javelle[3], Mehdi Mhalla[1,3], and Simon Perdrix[1,3]

[1] CNRS
[2] Institut Fourier, University of Grenoble, France
[3] LIG, University of Grenoble, France

Abstract. We study the graph-state-based quantum secret sharing protocols [24,17] which are not only very promising in terms of physical implementation, but also resource efficient since every player's share is composed of a single qubit. The threshold of a graph-state-based protocol admits a lower bound: for any graph of order n, the threshold of the corresponding n-player protocol is at least $0.506n$. Regarding the upper bound, lexicographic product of the C_5 graph (cycle of size 5) are known to provide n-player protocols which threshold is $n - n^{0.68}$. Using Paley graphs we improve this bound to $n - n^{0.71}$. Moreover, using probabilistic methods, we prove the existence of graphs which associated threshold is at most $0.811n$. Albeit non-constructive, probabilistic methods permit to prove that a random graph G of order n has a threshold at most $0.811n$ with high probability. However, verifying that the threshold of a given graph is acually smaller than $0.811n$ is hard since we prove that the corresponding decision problem is NP-Complete. These results are mainly based on the graphical characterization of the graph-state-based secret sharing properties, in particular we point out strong connections with domination with parity constraints.

Keywords: Quantum Information, Graph Theory, Quantum Cryptography, NP-Completeness.

1 Introduction

1.1 Quantum Secret Sharing

Secret sharing schemes were independently introduced by Shamir [33] and Blakley [3] and extended to the quantum case by Hillery [14] and Gottesman [8,10]. A quantum secret sharing protocol consists in encoding a secret into a multipartite quantum state. Each of the players of the protocol has a *share* which consists of a subpart of the quantum system and/or classical bits. *Authorized* sets of players are those that can recover the secret collectively using classical and quantum communications. A set of players is *forbidden* if they have no information about the secret. The encrypted secret can be a classical bit-string or a quantum state.

A *threshold* $((k, n))$ quantum secret sharing protocol [14,8,10] is a protocol by which a dealer distributes shares of a quantum secret to n players such that any subset of at least k players is authorized, while any set of less than k players is forbidden. It is assumed that the dealer has only one copy of the quantum secret he wants to share. A direct consequence of the no-cloning theorem [36] is that no $((k, n))$ quantum secret sharing protocol can exists when $k \leq \frac{n}{2}$ – otherwise two distinct sets of players can reconstruct the secret implying a cloning of the quantum secret. On the other hand, for any $k > \frac{n}{2}$ a $((k, n))$ protocol has been introduced in [8] in such a way that the dimension of each share is proportional to the number of players. The unbounded size of the share is a serious drawback of the protocol, as a consequence several schemes of quantum secret sharing using a bounded amount of resources for each player have been introduced [24,4,20].

1.2 Graph-State-Based Quantum Secret Sharing

In [24] a quantum secret sharing scheme using particular quantum states, called *graph states* [12], and such that every player receives a single qubit, has been introduced. The graph-state-based protocols are also of interest because graph states are at the forefront in terms of implementation and have emerged as a powerful and elegant family of entangled states [13,30].

As introduced in [24], only one non-trivial graph (the cycle of size 5) corresponds to a threshold protocol. In [17], the graph-state-based protocol has been extended to ensure that any graph correspond to a threshold protocol. Given a graph G, the threshold of the corresponding protocol is $\kappa_Q(G)$. This threshold is characterized graphically by the notion of weak odd domination. In [17], it has been proved that for any graph G of order n, $\kappa_Q(G) > 0.506n$, refining the no-cloning theorem. This bound is not known to be tight. All known constructions of graph-state-based quantum secret sharing protocols lead to quasi-uninimity protocols (i.e. the threshold is $n - o(n)$, where n is the number of players). The best known construction is based on the lexicographic product of graphs, and leads to protocols with a threshold $n - n^{0.68}$ [17].

We improve this threshold to $n-n^{0.71}$ using Paley graphs. Moreover, we show, using probabilistic methods, that for any (large enough) n there exists a graph G of order n such that $\kappa_Q(G) \leq 0.811n$. The proof is not constructive, but it crucially shows that graph-state based quantum secret sharing protocols are not restricted to quasi-unanimity thresholds. We actually prove that almost all the graphs have such a 'small' κ_Q: if one picks a random graph G of order n (every edge occurs with probability $1/2$), then $\kappa_Q(G) \leq 0.811n$ with probability greater than $1 - \frac{1}{n}$. We also prove that, given a graph G and a parameter k, deciding whether $\kappa_Q(G) \geq k$ is NP-complete. As a consequence, one cannot efficiently verify that a particular randomly generated graph has actually a 'small' κ_Q.

1.3 Combinatorial Properties of Graph States

The development and the study of graph-based protocols [24,20,17,31,16] have already pointed out deep connections between graph theory and quantum information theory. For instance, it has been shown [19] that a particular notion of flow [9,5,26,25] in the underlying graph captures the flow, during the protocol, of the information contained in the secret from the dealer – who encodes the secret and sends the shares – to the authorized sets of players. The results presented in this paper contribute to these deep connections: we show that weak odd domination is a key concept for studying the properties of graph-based quantum secret sharing protocols.

The study of graph-state-based protocols also contributes, as a by-product, to a better understanding of the combinatorial properties of the graph states. The graph state formalism is a very powerful tool which is used in several areas of quantum information processing. Graph states provide a universal resource for quantum computing [30,34,27] and are also used in quantum correction codes [32,6] for instance. They are also used to define pseudo-telepathy games [1]. Moreover, they are very promising in terms of physical implementation [29,35]. As a consequence, progresses in the knowledge of the fundamental properties of graph states can potentially impact not only quantum secret sharing but a wide area of quantum information processing.

2 Graph State Secret Sharing

2.1 Graph States

For a given graph $G = (V, E)$ with vertices $v_1, \dots, v_n$, the corresponding graph state $|G\rangle$ is a n-qubit quantum state defined as

$$|G\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{q(x)} |x\rangle$$

where $q(x)$ is the number of edges in the induced subgraph $G[x] = (\{v_i \in V \mid x_i = 1\}, \{(v_i, v_j) \in E \mid x_i = x_j = 1\})$.

Graph states have the following fundamental fixpoint property: given a graph G, for any vertex $u \in V$,

$$X_u Z_{N(u)} |G\rangle = |G\rangle$$

where $N(u)$ is the neighborhood of u in G, $X = |x\rangle \mapsto |\bar{x}\rangle$, $Z = |x\rangle \mapsto (-1)^x |x\rangle$ are one-qubit Pauli operators and $Z_A = \bigotimes_{u \in A} Z_u$ is a Pauli operator acting on the qubits in A.

2.2 Sharing a Classical Secret Using a Graph State

Graph-state-based classical sharing protocols have been introduced in [24]. In these protocols a classical secret is shared by means of a quantum state. The authorized sets of players are those which are satisfying the following graphical property, called *c-accessibility*:

Definition 1. *Given a graph $G = (V, E)$, a set $B \subseteq V$ of vertices is* c-accessible *if $\exists D \subseteq B$ such that $|D| = 1 \bmod 2$ and $Odd(D) \subseteq B$, where $Odd(D) := \{v \in V \mid |N(v) \cap D| = 1 \bmod 2\}$.*

Given a graph $G = (V, E)$ of order n, the **graph state-based protocol for sharing a classical secret** $s \in \{0, 1\}$ among n players is defined as:

1. **Encryption.** The dealer prepares the graph state $|G\rangle$. If $s = 1$ the dealer applies Z_V on the qubits of the graph state. The resulting state is
$$|G_s\rangle := Z_V^s |G\rangle$$
2. **Distribution.** Player j's share is qubit j of $|G_s\rangle$.
3. **Reconstruction.** Let B be a c-accessible set of player, with $D \subseteq B$ such that $|D| = 1 \bmod 2$ and $Odd(D) \subseteq B$.
   - The players in $Odd(D)$ apply $S = |x\rangle \mapsto i^x |x\rangle$ on their qubit.
   - The players in D apply $H = |x\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x |1\rangle)$ on their qubit.
   - Each player $j \in D \cup Odd(D)$ measures his qubit in the $\{|0\rangle, |1\rangle\}$-basis and broadcasts his result $s_i \in \{0, 1\}$ to the players in B.
   - The reconstructed secret is
$$|G_D| + \sum_{j \in D \cup Odd(D)} s_j \bmod 2$$
where $|G_D|$ is the size (i.e. the number of edges) of the subgraph induced by D.

Proof of the protocol [Sketch]. The proof that the reconstructed secret is the actual secret relies on the fact that for any $D \subseteq V$, $(-1)^{|G_D|} X_D Z_{Odd(D)} |G\rangle = |G\rangle$, as a consequence $(-1)^{|G_D|} X_D Z_{Odd(D)} |G_s\rangle = (-1)^s |G_s\rangle$, so a measurement according to $X_D Z_{Odd(D)}$ produces the classical outcome $s + |G_D| \bmod 2$. Finally, the non local measurement according to $X_D Z_{Odd(D)}$ can be decomposed into local measurements such that the parity of the local measurements is the same as the outcome of the non-local measurement. □

Sharing a classical bit can be done using a classical scheme, like [33], instead of using a quantum state. Moreover the above protocol can be simulated by purely classical graph-based schemes [16]. However, the study of the graph-state-based classical secret sharing, and in particular the characterization of their authorized structure are essential for the next sections where the sharing of a quantum secret is considered.

The graph-state-based classical secret sharing protocol is *perfect*, i.e. any set of players is either c-accessible, or has no information about the secret [17]. The proof of perfectness has two steps: using graph theory arguments, one can prove that if a set B of players is not c-accessible in a graph, then B is a WOD set (WOD stands for *weak odd domination*), i.e. there exists $C \subseteq V \setminus B$ such that $B \subseteq Odd(C)$. The second part of the proof consists in proving that the reduced density matrix of a WOD set of players does not depend on the secret, thus the players in this set have no information about the secret [19].

Since any subset of a WOD set is a WOD set and that any superset of a c-accessible is a c-accessible set, the important quanties for a graph state-based classical secret sharing protocol are the largest WOD set and the smallest c-accessible set by considering the following quantities:

Definition 2. *For a given graph G, let*

$$\kappa(G) = \max_{B \textit{ WOD}} |B| \qquad\qquad \kappa'(G) = \min_{B \textit{ c-accessible}} |B|$$

For instance, for a C_5 graph, i.e. a cycle of size 5, $\kappa(C_5) = 2$ and $\kappa'(C_5) = 3$. So it means that any set of at least 3 players can recover the secret whereas any set of at most 2 players have no information about the secret. In other words, the C_5 graph induces a threshold protocol. Another example is the complete graph K_n. Since $\kappa(K_n) = n - 1$ and $\kappa'(K_n) = n$, complete graphs induce unanimity protocol.

We consider a third example: for any $p, q \in \mathbb{N}$, let $G_{p,q}$ be the complete q-partite graph where each independent set is of size p: the vertices of $G_{p,q}$ are pairs (i, j), with $0 \leq i < p$, $0 \leq j < q$, two vertices (i, j), (i', j') are connected if and only if $j \neq j'$. $G_{p,q}$ is of order $n = pq$.

Lemma 1. *For any $p, q \in \mathbb{N}$,*

$$\begin{aligned} \kappa(G_{p,q}) = n - p \textit{ and } \kappa'(G_{p,q}) &= q & \textit{if } q = 1 \bmod 2 \\ \kappa(G_{p,q}) = \max(n - p, n - q) \textit{ and } \kappa'(G_{p,q}) &= p + q + 1 & \textit{if } q = 0 \bmod 2 \end{aligned}$$

Proof. If $q = 1 \bmod 2$

– $[\kappa(G_{p,q}) \geq n - p]$: The subset B composed of all the vertices but a maximal independent set (MIS) – i.e. an independent set of size p – is in the odd neighborhood of each vertex in $V \setminus B$. Therefore B is WOD and $|B| = n - p$. Consequently $\kappa(G_{p,q}) \geq n - p$.

– $[\kappa(G_{p,q}) \leq n - p]$: Any set B such that $|B| > n - p$ contains at least one vertex from each of the q MIS, i.e. a clique of size q. Let $D \subseteq B$ be such a clique of size $|D| = q = 1 \bmod 2$. Every vertex v outside D is connected to all the elements of D but the one in the same MIS as v. Thus $Odd(D) = \emptyset$. As a consequence, B is non-WOD.

– $[\kappa'(G_{p,q}) \leq q]$: B composed of one vertex from each MIS is a non-WOD set (see previous item).

– $[\kappa'(G_{p,q}) \geq q]$: If $|B| < q$ then B does not intersect all the MIS of size p, so B is in the odd neighborhood of each vertex of such a MIS. So B is WOD.

If $q = 0 \bmod 2$

– $[\kappa(G) \geq max(n - p, n - q)]$: For $\kappa(G) \geq n - p$, see previous lemma. The subset B composed of all the vertices but a clique of size q (one vertex from each MIS) is in the odd neighborhood of $V \setminus B$. Indeed each vertex of B is connected to $q - 1 = 1 \bmod 2$ vertices of $V \setminus B$. So B of size $n - q$ is WOD, as a consequence $\kappa(G) \geq n - q$.

– $[\kappa(G) \leq max(n-p, n-q)]$: Any set B such that $|B| > max(n-p, n-q)$ contains at least one vertex from each MIS and moreover it contains a MIS S of size q. Let $D \subseteq B \setminus S$ be a clique of size $q-1 = 1 \bmod 2$. Every vertex u in $V \setminus B$ is connected to all the vertices in D but one, so $Odd(D) \subseteq B$.

– $[\kappa'(G) \leq p+q-1]$: Let S be an MIS. Let B be the union of S and of a clique of size q. Let $D = B \setminus S$. $|D| = q-1 = 1 \bmod 2$. Every vertex u in $V \setminus B$ is connected to all the vertices of D but one, so $Odd(D) \subseteq B$.

– $[\kappa'(G) \geq p+q-1]$: Let $|B| < p+q-1$. If B does not intersect all the MIS of size p, then B is in the odd neighborhood of each vertex of such a non intersecting MIS. If B intersects all the MIS then it does not contain any MIS, thus there exists a clique $C \subseteq V \setminus B$ of size q. Every vertex in B is in the odd neighborhood of C. □

2.3 Sharing a Quantum Secret

Graph state based classical secret sharing can be extended to the quantum case as follows (the encryption method has been introduced in [24], and for the reconstruction method in [17]):

Given a graph G of order n, the **graph state-based protocol for sharing a quantum secret** $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ among n players is:

1. **Encryption.** The dealer prepares the quantum state $\alpha|G_0\rangle + \beta|G_1\rangle$ where $|G_0\rangle := |G\rangle$ and $|G_1\rangle := Z_V|G\rangle$.
2. **Distribution.** Player i's share is qubit i of $\alpha|G_0\rangle + \beta|G_1\rangle$.
3. **Reconstruction.** Let B be a c-accessible set of players such that $V \setminus B$ is WOD. So $\exists C, D \subseteq B$ such that $V \setminus B \subseteq Odd(C)$, $|D| = 1 \bmod 2$, and $Odd(D) \subseteq B$.
   – Players in B choose $u \in B$ who will reconstruct the secret. Every player in $B \setminus \{u\}$ sends his qubit to u.
   – u applies the unitary $\frac{1}{\sqrt{2}}\left(\begin{array}{c|c} I & U \\ \hline -U & I \end{array}\right)$ where $U = (-1)^{|G_D|}X_D Z_{Odd(D)}$ on the $(|B|+1)$-qubit system composed of an ancillary qubit $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and the received qubits. The resulting state of the $(n+1)$-qubit system is
   $$\alpha|0\rangle \otimes |G_0\rangle + \beta|1\rangle \otimes |G_1\rangle$$
   – u applies the unitary $\left(\begin{array}{c|c} I & 0 \\ \hline 0 & U' \end{array}\right)$ where $U' = (-1)^{|G_C|}X_C Z_{V\setminus Odd(C)}$. The resulting state of the $(n+1)$-qubit system is
   $$(\alpha|0\rangle + \beta|1\rangle) \otimes |G\rangle$$
   Thus the secret is reconstructed on the first qubit.

In this protocol of graph-state based quantum secret sharing, the authorized sets of players are c-accessible such that their complementary sets are WOD. Intuitively the c-accessibility allows the players to extend the superposition to an ancillary qubit (i.e. to transform the state $\alpha\left|G_0\right\rangle + \beta\left|G_1\right\rangle$ to $\alpha\left|0\right\rangle \otimes \left|G_0\right\rangle + \beta\left|1\right\rangle \otimes \left|G_1\right\rangle$). Notice that if the secret is classical (either α or β is equal to 0), then the state of the ancillary qubit is nothing but the secret. In the general case, when the secret is a superposition, the ancillary qubit is entangled with the rest of the system. The second requirement, namely that the complementary set $V \setminus B$ is WOD allows the players in B to make the ancillary qubit separable from the rest of system, producing the state $(\alpha\left|0\right\rangle + \beta\left|1\right\rangle) \otimes \left|G\right\rangle$.

Since authorized players are c-accessing such that their complementary are WOD, the important quantity for graph-state based quantum secret sharing protocols is

$$\kappa_Q(G) = \max(\kappa(G), n - \kappa'(G))$$

Indeed any set B of players such that $|B| > \kappa_Q(G)$ is c-accessing since $|B| > \kappa(G)$ and its complementary set is WOD since $|V \setminus B| < \kappa'(G)$.

2.4 Threshold Schemes

Notice that the graph-state based quantum secret sharing are not perfect in general. One can prove that the authorized sets are those which are c-accessible such that their complementary set is WOD [17]. On the otherhand, if a set is WOD and its complementary is c-accessible, then such a set has no information about the secret. But all sets which are not of that kind, for instance those such that both B and $V \setminus B$ are c-accessible, have some partial information about the secret.

To make the protocol perfect, and even to obtain a threshold protocol, a variant of the previous protocol has heen introduced in [17]. The idea is to add a one-time padding of the secret to ensure that the sets of players which size is below the threshold have no information about the secret. To implement this one-time padding the previous protocol is coupled with a classical protocol for sharing the classical key of the one-time padding.

Given a graph G and an integer $k > \kappa_Q(G)$ the **threshold graph state-based protocol for sharing a quantum secret** $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ among n players is:

1. **Encryption.** The dealer chooses uniformly at random two bits $p, q \in \{0,1\}$ and applies X^pZ^q to the secret. The resulting state is $\alpha|p\rangle + \beta(-1)^q|1-p\rangle$. Then, the dealer prepares the quantum state $\alpha|G_p\rangle + (-1)^q\beta|G_{1-p}\rangle$.
2. **Distribution.** Player i's share is qubit i of $\alpha|G_p\rangle + (-1)^q\beta|G_{1-p}\rangle$. Moreover p and q are shared among the n players using a classical secret sharing protocol (not described here) with threshold k.
3. **Reconstruction.** For any B such that $|B| \geq k$, since $|B| > \kappa_Q(G)$, $\exists C, D \subseteq B$ such that $V \setminus B \subseteq Odd(C)$, $|D| = 1 \bmod 2$, and $Odd(D) \subseteq B$.
   - Players in B choose $u \in B$ who will receive the secret. Every player in $B \setminus \{u\}$ sends his qubit to u.
   - u applies $\frac{1}{\sqrt{2}}\left(\begin{array}{c|c} I & U \\ \hline -U & I \end{array}\right)$ where $U = (-1)^{|G_D|}X_DZ_{Odd(D)}$ and then $\left(\begin{array}{c|c} I & 0 \\ \hline 0 & U' \end{array}\right)$ where $U' = (-1)^{|G_C|}X_CZ_{V\setminus Odd(C)}$. The resulting state is
   $$(\alpha|p\rangle + \beta(-1)^q|1-p\rangle) \otimes |G\rangle$$
   - Using the classical secret sharing protocol, the players in B reconstructs the classical bits p and q.
   - u applies Z^qX^p to the ancillary qubit. The resulting state is $\alpha|0\rangle + \beta|1\rangle$.

This protocol is a $((k,n))$ threshold quantum secret sharing protocol. Indeed for any set of at most $k-1$ players, they have no information about the classical keys p and q which garantees that they have no information about the quantum secret. For the sets of at least k players, since $k > \kappa_Q(G)$, they can reconstruct the state $\alpha|p\rangle + \beta(-1)^q|1-p\rangle$, like in the previous protocol, moreover they have access to the keys p and q, so they can reconstruct the quantum secret.

2.5 Lower Bound and Quasi-unanimity Protocols

Graph-state-based secret sharing protocols are very promising in terms of physical implementations [29,35], moreover, contrary to the family of quantum secret sharing introduced by Gottesman [10], the size of each quantum share does not depend on the number of players. The drawback is that some threshold protocols cannot be implemented using a graph-state-based protocol:

Lemma 2 (Lower bound [17]). *For any graph G of order $n > 5$,*

$$\kappa_Q(G) > 0.506n$$

Notice that for any quantum secret sharing scheme, if there is a threshold, this threshold must be larger than $n/2$. This bound [10] is a direct application of the no-cloning theorem: if $k < n/2$ then two distinct sets of k players can recover the quantum secret, leading to a cloning of the quantum secret. The lower bound for graph-state-based protocols is not known to be tight.

Regarding the upper bound, all known constructions of graph-state-based quantum secret sharing protocols are quasi-unanimity protocols. In the next section, we will prove using non constructive methods that for any n there exists a graph G of order n such that $\kappa_Q(G) < 0.811n$.

Thanks to lemma 1, for any $p, q \in \mathbb{N}$, the complete q-partite graph $G_{p,q}$ of order $n = pq$ (see section 2.2), $\kappa_Q(G_{p,q}) = \max(n-p, n-q)$. Thus, for any square number n, $\kappa_Q(G_{\sqrt{n},\sqrt{n}}) = n - \sqrt{n}$. The corresponding secret sharing protocol is a quasi-unanimity protocol since the ratio k/n tends to 1 as n tends to infinity.

The best known construction is based on the lexicographic product of graphs: Given $G = (V, E)$ $G \bullet G = (V', E')$ is defined as $V' := V \times V$ and $E' := \{((u_1, u_2), (v_1, v_2)) \mid (u_1, v_1) \in E \text{ or } (u_1 = v_1 \wedge (u_2, v_2) \in E)\}$.

Lemma 3 ([17]). *For any graph G, of order n*

$$\kappa_Q(G \bullet G) \leq 2n.\kappa_Q(G) - \kappa_Q(G)^2$$

Graphs-state-based quantum secret sharing protocols with threshold $n - n^{0.68}$, where n is the number of players, can be obtained using inductively the lexicographic product of C_5 graph (cycle on five vertices) [17].

In fact, the C_5 graph is a particular case of a family of graphs called Paley graphs, that have been used recently in [18] to provide the best known upper bound for the minimum degree up to local complementation which can be defined as $\min_{D \subseteq V, D \neq \emptyset} |D \cup Odd(D)| - 1$.

For any prime p such that $p = 1 \bmod 4$, the Paley graph Pal_p is a graph on p vertices where each vertex is an element of $\mathbb{F}_p$. There is an edge between two vertices i and j if and only if $i - j$ is a square in $\mathbb{F}_p$.

Graphs-state-based quantum secret sharing protocols with n players and threshold $n - n^{0.71}$ is obtained using the lexicographic product of Pal_{29} graphs. This is, up to our knowledge, the best known constructive threshold.

Theorem 1. *For any $i > 0$,*

$$\kappa_Q({Pal_{29}}^{\bullet^i}) \leq n - n^{\frac{\log(11)}{\log(29)}} \approx n - n^{0.71}$$

where ${Pal_{29}}^{\bullet^1} = Pal_{29}$, ${Pal_{29}}^{\bullet^i} = {Pal_{29}}^{\bullet^{i-1}} \bullet {Pal_{29}}^{\bullet^{i-1}}$ and $n = 29^{2^i}$ is the order of the graph.

Proof. $\kappa_Q(Pal_{29})$ is computed taking benefits of the symmetries of the Paley graphs (strong regularity, vertex transitivity, edge transitivity, self complementarity). The evolution with the lexicographic product is given by lemma 3. □

It is significant and interesting to notice that the conjecture of the existence of an infinite family of Paley graphs leading to non quasi-unanimity protocols is related to the Bazzi-Mitter conjecture [2].

3 Graphs with Small κ_Q

In this section, we prove the existence of graph-state-based secret sharing protocols which are not quasi-unanimity. More precisely, using the asymmetric Lovász Local Lemma [23] we show that there exists an infinite family of graphs $\{G_i\}$ such that $\kappa_Q(G_i) \leq 0.811 n_i$ where n_i is the order of G_i. Moreover, we prove that a random graph $G(n, 1/2)$ (graph on n vertices where each pair of vertices have probability 1/2 to have an edge connecting them) satisfies $\kappa_Q(G(n, 1/2)) \leq 0.811n$ with high probability.

First we prove the following lemma:

Lemma 4. *Given k and $G = (V, E)$, if for any non empty set $D \subseteq V$, $|D \cup Odd(D)| > n - k$ and $|D \cup (V \setminus Odd(D))| > n - k$ then $\kappa_Q(G) < k$.*

Proof. Since $\forall D \subseteq V$ $|D \cup Odd(D)| > n-k$, $\kappa'(G) > n-k$. Let $B \subseteq V$, $|B| \geq k$, if B is not WOD then $\exists C \subseteq V \setminus B$ such that $B \subseteq Odd(C)$, so $(V \setminus Odd(C)) \subseteq V \setminus B$ which implies $|C \cup (V \setminus Odd(C))| \leq n - k$. $\square$

We use the asymmetric form of the Lovász Local Lemma that can be stated as follows:

Theorem 2 (Asymmetric Lovász Local Lemma, no independency case). *Let $\mathcal{A} = \{A_1, \cdots, A_n\}$ be a set of bad events in an arbitrary probability space. If for every A_i there exists $w(A_i) \in [0, 1)$ such that $Pr(A_i) \leq w(A_i).p$, where $p = \prod_{A_j \in \mathcal{A}}(1 - w(A_j))$ then $Pr(\overline{A_1}, \cdots, \overline{A_n}) \geq p$*

Theorem 3. *There exists an infinite family of graphs $\{G_i\}$ such that $\kappa_Q(G_i) \leq 0.811 n_i$ where n_i is the order of G_i.*

Proof. Let $G(n, 1/2) = (V, E)$ be a random graph. We use the asymmetric Lovász local lemma to show that the probability that for all non empty set $D \subseteq V$ $|D \cup Odd(D)| > (1 - c)n$ and $|D \cup (V \setminus Odd(D))| > (1 - c)n$ is positive for some constant c. This ensures by Lemma 4 that $\kappa_Q(G) < cn$.

We consider the "bad" events A_D : $|Odd(D) \cup D| \leq (1 - c)n$ and A'_D : $|Odd(D) \cup (V \setminus Odd(D))| \leq (1-c)n$. When $|D| > (1-c)n$, $Pr(A_D) = Pr(A'_D) = 0$, therefore the previous events are defined for all D such that $|D| \leq (1 - c)n$.

For all D such that $|D| \leq (1-c)n$, we want to get an upper bound on $Pr(A_D)$. Let $|D| = dn$ for some $d \in (0, 1 - c]$. For all $u \in V \setminus D$, $Pr(\text{“}u \in Odd(D)\text{”}) = \frac{1}{2}$. If D is fixed, the events "$u \in Odd(D)$" when u is outside D are independent. Therefore, if the event A_D is true, any but at most $(1-c-d)n$ vertices outside D are contained in $Odd(D)$. There are $(1-d)n$ vertices outside D, then $Pr(A_D) = \left(\frac{1}{2}\right)^{(1-d)n} \sum_{k=0}^{(1-c-d)n} \binom{(1-d)n}{k} \leq \left(\frac{1}{2}\right)^{(1-d)n} 2^{(1-d)nH\left(\frac{1-c-d}{1-d}\right)} = 2^{(1-d)n\left[H\left(\frac{c}{1-d}\right)-1\right]}$ where $H : t \mapsto -t\log_2(t) - (1-t)\log_2(1-t)$ is the binary entropy function. Similarily, $Pr(A'_D) \leq 2^{(1-d)n\left[H\left(\frac{c}{1-d}\right)-1\right]}$.

We consider that all the events can be dependent. For any $D \subseteq V$ such that $0 < |D| \leq (1-c)n$, we define $w(A_D) = w(A'_D) = \frac{1}{r\binom{n}{|D|}}$. First, we verify that $Pr(A_D) \leq w(A_D) \prod_{D' \subseteq V, |D| \leq (1-c)n}(1 - w(A_{D'}))(1 - w(A'_{D'}))$.

The product of the right-hand side of the previous equation can be written $p = \prod_{|D'|=1}^{(1-c)n} \left(1 - \frac{1}{r\binom{n}{|D'|}}\right)^{2\binom{n}{|D'|}} = \left[\prod_{|D'|=1}^{(1-c)n} \left(1 - \frac{1}{r\binom{n}{|D'|}}\right)^{r\binom{n}{|D'|}}\right]^{\frac{2}{r}}$. The function $f : x \mapsto \left(1 - \frac{1}{x}\right)^x$ verifies $f(x) \geq \frac{1}{4}$ when $x \geq 2$, therefore $p \geq \left(\frac{1}{4}\right)^{\frac{2}{r}(1-c)n} = 2^{-\frac{4(1-c)n}{r}}$ for any $r \geq 2$. Thus, it is sufficient to have $2^{(1-d)n\left[H\left(\frac{c}{1-d}\right)-1\right]} \leq \frac{1}{r\binom{n}{dn}} 2^{-\frac{4(1-c)n}{r}}$. Rewriting this inequality gives $r\binom{n}{dn} 2^{(1-d)n\left[H\left(\frac{c}{1-d}\right)-1\right]+\frac{4(1-c)n}{r}} \leq 1$. Thanks to the bound $\binom{n}{dn} \leq 2^{nH\left(\frac{dn}{n}\right)}$ and after applying the logarithm function and dividing by n, it is sufficient that $(1-d)\left[H\left(\frac{c}{1-d}\right) - 1\right] + H(d) + \frac{4(1-c)}{r} + \frac{\log_2 r}{n} \leq 0$. If we take $r = n$, the condition becomes asymptotically $(1-d)\left[H\left(\frac{c}{1-d}\right) - 1\right] + H(d) \leq 0$.

Numerical analysis shows that this condition is true for any $c > 0.811$ and for all $d \in (0, 1-c]$. Thus, thanks to the Lovász Local Lemma, for any $c > 0.811$, $Pr(\kappa_Q(G) < cn) \geq p \geq \left(\frac{1}{4}\right)^{\frac{2}{r}(1-c)n} > 0$, therefore there exists an infinite family of graphs $\{G_i\}$ such that $\kappa_Q(G_i) \leq 0.811 n_i$ where n_i is the order of G_i for $n_i \geq N_0$ for some $N_0 \in \mathbb{N}$. □

Recently [31], Sarvepalli proved that quantum secret sharing protocols based on graph states are equivalent to quantum codes. Combining this result with the Gilbert Varshamov bounds on quantum stabilizer codes [11], we can provide an alternative proof of theorem 3. However, we believe the use of the Lovász Local Lemma offers several advantages: the proof is a purely graphical proof with a potential extension to the construction of good quantum secret sharing schemes using the recent development in the algorithmic version [28] of the Lovász Local Lemma. Moreover, the use of the probabilistic methods already offers a way of generating good quantum secret sharing protocols with high probability by adjusting the parameters of the Lovász Local Lemma:

Theorem 4. *There exists n_0 such that for any $n > n_0$, a random graph $G(n, \frac{1}{2})$ has a κ_Q smaller than $0.811n$ with high probability:*

$$Pr\left(\kappa_Q(G(n, \frac{1}{2})) < 0.811n\right) \geq 1 - \frac{1}{n}$$

Proof. The proof of the theorem is done as in the proof of theorem 3, by taking $c = 0.811$ and $r = 4\ln(2)(1-c)n^2$. It guarantees that for $n \geq 26681$, $(1-d)\left[H\left(\frac{c}{1-d}\right) - 1\right] + H(d) + \frac{4(1-c)}{r} + \frac{\log_2 r}{n} \leq 0$. Thus, for any $D \subseteq V$ such that $0 < |D| \leq (1-c)n$, $Pr(A_D) \leq w(A_D)\prod_{D' \subseteq V, |D| \leq (1-c)n}(1 - w(A_{D'}))(1 - w(A'_{D'}))$. Moreover the probability that none of the bad events occur is $Pr\left(\kappa_Q(G(n, \frac{1}{2})) < 0.811n\right) \geq \left(\frac{1}{4}\right)^{\frac{2}{r}(1-c)n} = \left(\frac{1}{4}\right)^{\frac{1}{2n\ln(2)}} = e^{-\frac{1}{n}} \geq 1 - \frac{1}{n}$.

□

4 Complexity of Computing the Threshold of Graph-State-Based Protocols

According to Theorem 4, a random graph $G(n, 1/2)$ induces a secret sharing protocol with a threshold smaller than $0.811n$ with high probability: $Pr\left(\kappa_Q(G(n, \frac{1}{2})) < 0.811n\right) \geq 1 - \frac{1}{n}$. So even if the Lovász local Lemma is not constructive, one can pick uniformly at random a graph G of order n, if $\kappa_Q(G) \geq 0.811n$, he picks another one and so on. Since the probability that $\kappa_Q(G) \geq 0.811n$ this procedure seems to be efficient, however the crucial point here is the complexity of deciding whether $\kappa_Q(G) \geq 0.811n$ or not. In this section we consider the complexity of this problem and show that the problem is NP-complete (Theorem 10). To prove this result we introduce several bounds and complexity results on weak odd domination, and in particular on the quantities $\kappa(G)$, $\kappa'(G)$ and $\kappa_Q(G)$ of a graph G.

First, we show that the sum of $\kappa(G)$ and $\kappa'(\overline{G})$ is always greater than the order of the graph G. The proof is based on the duality property that the complement of a non-WOD set in G is a WOD set in $\overline{G}$, the complement graph of G.

Lemma 5. *Given a graph $G = (V, E)$, if $B \subseteq V$ is not a WOD set in G then $V \setminus B$ is a WOD set in $\overline{G}$.*

Proof. Let B be a non-WOD set in G. $\exists D \subseteq B$ such that $|D| = 1 \bmod 2$ and $Odd_G(D) \subseteq B$. As a consequence, $\forall v \in V \setminus B$, $|N_G(v) \cap D| = 0 \bmod 2$. Since $|D| = 1 \bmod 2$, $\forall v \in V \setminus B$, $|N_{\overline{G}}(v) \cap D| = 1 \bmod 2$. Thus, $V \setminus B$ is a WOD set in $\overline{G}$. □

Theorem 5. *For any graph G of order n, $\kappa'(G) + \kappa(\overline{G}) \geq n$.*

Proof. There exists a non-WOD set $B \subseteq V$ such that $|B| = \kappa'(G)$. According to Lemma 5, $V \setminus B$ is WOD in $\overline{G}$, so $n - |B| \leq \kappa(\overline{G})$, so $n - \kappa'(G) \leq \kappa(\overline{G})$. □

For any vertex v of a graph G, its (open) neighborhood $N(v)$ is a WOD set, whereas its closed neighborhood (i.e. $N[v] = \{v\} \cup N(v)$) is a non-WOD set, as a consequence:

$$\kappa(G) \geq \Delta \qquad\qquad \kappa'(G) \leq \delta + 1$$

where Δ (resp. δ) denotes the maximal (resp. minimal) degree of the graph G.

In the following, we prove an upper bound on $\kappa(G)$ and a lower bound on $\kappa'(G)$.

Lemma 6. *For any graph G of order n and degree Δ, $\kappa(G) \leq \frac{n.\Delta}{\Delta+1}$.*

Proof. Let $B \subseteq V$ be a WOD set. $\exists C \subseteq V \setminus B$ such that $B \subseteq Odd(C)$. $|C| \leq n - |B|$ and $|B| \leq |Odd(C)| \leq \Delta.|C|$, so $|B| \leq \Delta.(n - |B|)$. It comes that $|B| \leq \frac{n.\Delta}{\Delta+1}$, so $\kappa(G) \leq \frac{n.\Delta}{\Delta+1}$. □

In the following we prove that this bound is reached only for graphs having a perfect code. A graph $G = (V, E)$ has a perfect code if there exists $C \subseteq V$ such that C is an independent set and every vertex in $V \setminus C$ has exactly one neighbor in C.

Theorem 6. *For any graph G of order n and degree Δ, $\kappa(G) = \frac{n.\Delta}{\Delta+1}$ if and only if G has a perfect code C such that $\forall v \in C$, $d(v) = \Delta$.*

Proof. ($\Leftarrow$) Let C be a perfect code of G such that $\forall v \in C$, $\delta(v) = \Delta$. $V \setminus C$ is a WOD set since $Odd(C) = V \setminus C$. Moreover $|V \setminus C| = \frac{n\Delta}{\Delta+1}$, so $\kappa(G) \geq \frac{n.\Delta}{\Delta+1}$. According to Lemma 6, $\kappa(G) \leq \frac{n\Delta}{\Delta+1}$, so $\kappa(G) = \frac{n\Delta}{\Delta+1}$.
($\Rightarrow$) Let B be a WOD set of size $\frac{n.\Delta}{\Delta+1}$. There exists $C \subseteq V \setminus B$ such that $B \subseteq Odd(C)$. Notice that $|C| \leq n - \frac{n.\Delta}{\Delta+1} = \frac{n}{\Delta+1}$. Moreover $|C|.\Delta \geq |Odd(C)| \geq |B|$, so $|C| = \frac{n}{\Delta+1}$. It comes that $|B| = |B \cap Odd(C)| \leq \sum_{v \in C} d(v) \leq \Delta.\frac{n}{\Delta+1} = |B|$. Notice that if C is not a perfect code the first inequality is strict, and if $\exists v \in C$, $d(v) < \Delta$, the second inequality is strict. Consequently, C is a perfect code and $\forall v \in C$, $d(v) = \Delta$. □

Corollary 1. *Given a Δ-regular graph G, $\kappa(G) = \frac{n\Delta}{\Delta+1}$ if and only if G has a perfect code.*

We consider the problem MaxWOD which consists in deciding, given a graph G and an integer $k \geq 0$, whether $\kappa(G) \geq k$.

Theorem 7. MaxWOD *is NP-Complete.*

Proof. MaxWOD is in the class NP since a WOD set B of size k is a YES certificate. Indeed, deciding whether the certificate B is WOD or not can be done in polynomial time by solving for X the linear equation $\Gamma_{V \setminus B}.X = 1_B$ in $\mathbb{F}_2$, where 1_B is a column vector of dimension $|B|$ where all entries are 1, and $\Gamma_{V \setminus B}$ is the cut matrix, i.e. a submatrix of the adjacency matrix of the graph which columns correspond to the vertices in $V \setminus B$ and rows to those in B. In fact, $X \subseteq V \setminus B$ satisfies $\Gamma_{V \setminus B}.X = 1_B$ if and only if ($X \subseteq V \setminus B$ and $B \subseteq Odd(X)$) if and only if B is WOD. For the completeness, given a 3-regular graph, if $\kappa(G) \geq \frac{3}{4}n$ then $\kappa(G) = \frac{3}{4}n$ (since $\kappa(G) \leq \frac{n\Delta}{\Delta+1}$ for any graph). Moreover, according to Corollary 1, $\kappa(G) = \frac{3}{4}n$ if and only if G has a perfect code. Since the problem of deciding whether a 3-regular graph has a perfect code is known to be NP complete (see [22] and [21]), so is MaxWOD. □

Now we introduce a lower bound on κ'.

Lemma 7. *For any graph G, $\kappa'(G) \geq \frac{n}{n-\delta}$ where δ is the minimal degree of G.*

Proof. According to Theorem 5, $\kappa'(G) \geq n - \kappa(\overline{G})$. Moreover, thanks to Lemma 6, $n - \kappa(\overline{G}) \geq n - \frac{n\Delta(\overline{G})}{\Delta(\overline{G})+1} = n - \frac{n(n-1-\delta(G))}{n-\delta(G)} = \frac{n}{n-\delta}$. □

This bound is reached for the regular graphs for which their complement graph has a perfect code, more precisely:

Theorem 8. *Given G a δ-regular graph such that $\frac{n}{n-\delta}$ is odd, $\kappa'(G) = \frac{n}{n-\delta}$ if and only if $\overline{G}$ has a perfect code.*

Proof. ($\Leftarrow$) Let C be a perfect code of $\overline{G}$. Since $|C| = \frac{n}{\Delta(\overline{G})+1} = \frac{n}{n-\delta} = 1 \mod 2$, $Odd_G(C) \subseteq C$, thus C is a non-WOD set in G, so $\kappa'(G) \leq \frac{n}{n-\delta}$. Since $\kappa'(G) \geq \frac{n}{n-\delta}$ for any graph, $\kappa'(G) = \frac{n}{n-\delta}$
($\Rightarrow$) Let B be a non-WOD set of size $\frac{n}{n-\delta}$ in G. $\exists D \subseteq B$ such that $|D| = 1 \bmod 2$ and $Odd_G(D) \subseteq B$. According to Lemma 5, $V \setminus B \subseteq Odd_{\overline{G}}(D)$, so $|Odd_{\overline{G}}(D)| \geq \Delta(\overline{G})\frac{n}{n-\delta}$, which implies that $|D|.\Delta(\overline{G}) \geq \Delta(\overline{G})\frac{n}{n-\delta}$. As a consequence, $|D| = \frac{n}{n-\delta}$ and since every vertex of $V \setminus B$ (of size $\Delta(\overline{G})\frac{n}{n-\delta}$) in $\overline{G}$ is connected to D, D must be a perfect code. □

We consider the problem MIN¬WOD which consists in deciding, given a graph G and an integer $k \geq 0$, whether $\kappa'(G) \leq k$?

Theorem 9. MIN¬WOD *is NP-Complete.*

Proof. MIN¬WOD is in the class NP since a non-WOD set of size k is a YES certificate. For the completeness, given a 3-regular graph G, if $\frac{n}{4}$ is odd then according to Theorem 8, G has a perfect code if and only if $\kappa'(\overline{G}) = \frac{n}{4}$. If $\frac{n}{4}$ is even, we add a K_4 gadget to the graph G. Indeed, $G \cup K_4$ is a 3-regular graph and $\frac{n+4}{4} = \frac{n}{4} + 1$ is odd. Moreover, G has a perfect code if and only if $G \cup K_4$ has a perfect code if and only if $\kappa'(\overline{G \cup K_4}) = \frac{n}{4} + 1$. Since deciding whether a 3-regular graph has a perfect code is known to be NP complete, so is MIN¬WOD

□

In the following, we prove that deciding, given a graph G and $k \geq 0$, whether $\kappa_Q(G) \geq k$ is NP complete (Theorem 10). The proof consists in a reduction from the problem MIN¬WOD, which is based on the evaluation of κ and κ' for particular graphs consisting of multiple copies of a same graph:

Lemma 8. *For any graph G and any $r > 0$, $\kappa(G^r) = r.\kappa(G)$ and $\kappa'(G^r) = \kappa'(G)$ where $G^1 = G$ and $G^{r+1} = G \cup G^r$.*

Proof.
– [$\kappa(G^r) = r.\kappa(G)$]: Let B be a WOD set in G of size $\kappa(G)$. B is in the odd neighborhood of some $C \subseteq V$. Then the set $B_r \subseteq V(G^r)$ which is the union of sets B in each copy of the graph G is in the odd neighborhood of $C_r \subseteq V(G^r)$, the union of sets C of each copy of G. Therefore B_r is WOD and $\kappa(G^r) \geq r.\kappa(G)$. Now if we pick any set $B_0 \subseteq V(G^r)$ verifying $|B_0| > r.\kappa(G)$, there exists a copy of G such that $|B_0 \cap G| > \kappa(G)$. Therefore B_0 is a non-WOD set and $\kappa(G^r) \leq r.\kappa(G)$.
– [$\kappa'(G^r) = \kappa'(G)$]: Let B be a non-WOD set in G of size $\kappa'(G)$. If we consider B as a subset of $V(G^r)$ contained in one copy of the graph G, B is a non-WOD set in G^r. Therefore $\kappa'(G^r) \leq \kappa'(G)$. If we pick any set $B \subseteq V(G^r)$ verifying $|B| < \kappa'(G)$, its intersection with each copy of G verifies $|B \cap G| < \kappa'(G)$. Thus, each such intersection is in the odd neighborhood of some C_i. So B is in the odd neighborhood of $\bigcup_{i=1..r} C_i$. Consequently, B_0 is a WOD set in G^r and $\kappa'(G^r) \geq \kappa'(G)$. □

We consider the problem QuantumThreshold which consists in deciding, for a given graph G and $k \geq 0$, whether $\kappa_Q(G) \geq k$, i.e. $\kappa(G) \geq k$ or $\kappa'(G) \leq n-k$?

Theorem 10. QuantumThreshold *is NP-Complete.*

Proof. QuantumThreshold is in NP since a WOD set of size k or a non-WOD set of size $n-k$ is a YES certificate. For the completeness, we use a reduction to the problem Min¬WOD. Given a graph G and any $k \geq 0$, $\kappa_Q(G^{k+1}) \geq (k+1)n-k \Leftrightarrow \Big(\kappa(G^{k+1}) \geq (k+1)n-k$ or $\kappa'(G^{k+1}) \leq k\Big) \Leftrightarrow \Big(\kappa(G) \geq n-1+\frac{1}{k+1}$ or $\kappa'(G) \geq k\Big) \Leftrightarrow \Big(\kappa(G) > n-1$ or $\kappa'(G) \geq k\Big)$. In the last disjunction, the first inequality $\kappa(G) > n-1$ is always false since for any graph G of order n we have $\kappa(G) \leq n-1$. Thus, the answer of the oracle call gives the truth of the second inequality $\kappa'(G) \geq k$ which corresponds to the problem Min¬WOD. As a consequence, QuantumThreshold is NP-complete. □

5 Conclusion

In this paper, we have studied secret sharing with graph states which lead to the analysis of the combinatorial quantity κ_Q that can be computed on graphs. We have studied and computed these quantities on some specific families of graphs, providing the best known constructive threshold protocols for graph-state based secret sharing. Then, we have proven using probabilistic methods that there exist graphs that allow non-quasi-unanimity protocols and that a random graph G of order n satisfies $\kappa_Q(G) \leq 0.811n$ with high probability. Finally, we have shown that given a graph G and an integer k, deciding whether $\kappa_Q(G) \geq k$ is NP-Complete. Recently, in [7], the analysis of this problem has been refined by considering its parameterized complexity: the problem belongs to W[2] and is hard for W[1].

It is very interesting to see that the best known protocols use Paley graph states and that they seem promising candidates to have even better bounds. Paley graph states have also been used in [18] to provide the best known family in terms of minimum degree up to local complementation, which is related to the complexity of graph state preparation [15]. Paley graph states also form an optimal family in terms of multipartie nonlocality [1]. Thus, these states seem very interesting in terms of entanglement and might be useful for other applications in quantum information theory.

Acknowledgements. This work has been funded by the ANR-10-JCJC-0208 CausaQ grant.

References

1. Anshu, A., Mhalla, M.: Pseudo-telepathy games and genuine NS n-way nonlocality using graph states arxiv:1207.2276
2. Bazzi, L.M.J., Mitter, S.K.: Some randomized code constructions from group actions. IEEE Transactions on Information Theory 52(7), 3210–3219 (2006)
3. Blakley, G.R.: Safeguarding cryptographic keys. In: AFIPS Conference Proceedings, vol. 48, pp. 313–317 (1979)
4. Broadbent, A., Chouha, P.R., Tapp, A.: The GHZ state in secret sharing and entanglement simulation. arXiv:0810.0259 (2008)
5. Browne, D.E., Kashe, E., Mhalla, M., Perdrix, S.: Generalized ow and determinism in measurement-based quantum computation. New Journal of Physics (NJP) 9(8) (2007)
6. Beigi, S., Chuang, I., Grassl, M., Shor, P., Zeng, B.: Graph concatenation for quantum codes. Journal of Mathematical Physics 52(2), 022201 (2011)
7. Cattanéo, D., Perdrix, S.: Parametrized Complexity of Weak Odd Domination Problems. arXiv:1206.4081 (2012)
8. Cleve, R., Gottesman, D., Lo, H.-K.: How to Share a Quantum Secret. Phys. Rev. Lett. 83, 648–651 (1999)
9. Danos, V., Kashe, E.: Determinism in the one-way model. Physical Review A 74(052310) (2006)
10. Gottesman, D.: Theory of quantum secret sharing. Phys. Rev. A 61, 042311 (2000)
11. Feng, K., Ma, Z.: A finite Gilbert-Varshamov bound for pure stabilizer quantum codes. IEEE Transactions on Information Theory 50, 3323–3325 (2004)
12. Hein, M., Dür, W., Eisert, J., Raussendorf, R., Van den Nest, M., Briegel, H.J.: Entanglement in graph states and its applications. In: Proceedings of the International School of Physics "Enrico Fermi" on "Quantum Computers, Algorithms and Chaos" (2005)
13. Hein, M., Eisert, J., Briegel, H.J.: Multi-party entanglement in graph states. Physical Review A 69, 062311, quant-ph/0307130 (2004)
14. Hillery, M., Buzek, V., Berthiaume, A.: Quantum Secret Sharing. Phys. Rev. A 59, 1829, arXiv/9806063 (1999)
15. Høyer, P., Mhalla, M., Perdrix, S.: Resources required for preparing graph states. In: 17th International Symposium on Algorithms and Computation (2006)
16. Javelle, J., Mhalla, M., Perdrix, S.: Classical versus Quantum Graph-based Secret Sharing. eprint:arXiv:1109.4731 (2011)
17. Javelle, J., Mhalla, M., Perdrix, S.: New protocols and lower bound for quantum secret sharing with graph states. In: Theory of Quantum Computation, Communication and Cryptography (TQC 2012). LNCS (to appear, 2012) eprint:arXiv:1109.1487
18. Javelle, J., Mhalla, M., Perdrix, S.: On the Minimum Degree Up to Local Complementation: Bounds and Complexity. In: Golumbic, M.C., Stern, M., Levy, A., Morgenstern, G. (eds.) WG 2012. LNCS, vol. 7551, pp. 138–147. Springer, Heidelberg (2012)
19. Kashefi, E., Markham, D., Mhalla, M., Perdrix, S.: Information flow in secret sharing protocols. EPTCS 9, 87–97 (2009)
20. Keet, A., Fortescue, B., Markham, D., Sanders, B.C.: Quantum secret sharing with qudit graph states. Phys. Rev. A 82, 062315 (2010)
21. Klavzar, S., Milutinovic, U., Petr, C.: 1-perfect codes in sierpinski graphs. Bulletin of the Australian Mathematical Society 66, 369–384 (2002)

22. Kratochvil, J.: Perfect codes in general graphs. In: 7th Hungarian colloqium on combinatorics, Eger (1987)
23. Lovász, L.: Problems and results on 3-chromatic hypergraphs and some related questions. In: Colloquia Mathematica Societatis Janos Bolyai, pp. 609–627 (1975)
24. Markham, D., Sanders, B.C.: Graph states for quantum secret sharing. Physical Review A 78, 042309 (2008)
25. Mhalla, M., Murao, M., Perdrix, S., Someya, M., Turner, P.: Which graph states are useful for quantum information processing? In: Theory of Quantum Computation, Communication and Cryptography (TQC 2011). LNCS (2011) (to appear)
26. Mhalla, M., Perdrix, S.: Finding Optimal Flows Efficiently. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 857–868. Springer, Heidelberg (2008)
27. Mhalla, M., Perdrix, S.: Graph States, Pivot Minor, and Universality of (X,Z)-measurements. International Journal of Unconventional Computing (to be published, 2012)
28. Moser, R.A., Tardos, G.: A constructive proof of the general Lovász Local Lemma. Journal of the ACM (JACM) 57(2), 1–15 (2010)
29. Prevedel, R., Walther, P., Tiefenbacher, F., Bohi, P., Kaltenbaek, R., Jennewein, T., Zeilinger, A.: High-speed linear optics quantum computing using active feed-forward. Nature 445(7123), 65–69 (2007)
30. Raussendorf, R., Briegel, H.J.: A one-way quantum computer. Physical Review Letters 86(22), 5188–5191 (2001)
31. Sarvepalli, P.: Non-Threshold Quantum Secret Sharing Schemes in the Graph State Formalism eprint:arXiv:1202.3433 (2012)
32. Schlingemann, D., Werner, R.F.: Quantum error-correcting codes associated with graphs. Phys. Rev. A 65, 012308 (2001)
33. Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)
34. Van den Nest, M., Miyake, A., Dür, W., Briegel, H.J.: Universal resources for measurement-based quantum computation. Phys. Rev. Lett. 97, 150504 (2006)
35. Walther, P., Resch, K.J., Rudolph, T., Schenck, E., Weinfurter, H., Vedral, V., Aspelmeyer, M., Zeilinger, A.: Experimental one-way quantum computing. Nature 434(7030), 169–176 (2005)
36. Wootters, W.K., Zurek, W.H.: A single quantum cannot be cloned. Nature 299, 802–803 (1982)

Testing Embedded Memories: A Survey

Said Hamdioui

Computer Engineering Lab
Delft Univeristy of Technology
Mekelweg 4, 2628CD, Delft
The Netherlands
S.Hamdioui@tudelft.nl
http://www.ce.ewi.tudelft.nl/hamdioui/

Abstract. According to the International Technology Roadmap for Semiconductors, embedded memories will continue to dominate the increasing system on chips (SoCs) content in the future, approaching 90% in in some cases. Therefore, the memory yield and quality will have a dramatic impact on the overall SoC cost and outgoing product quality. Meeting a high memory yield and quality requires understanding memory designs, modeling their faulty behaviors in appropriate and accurate way, designing adequate tests and diagnosis strategies as well as efficient Design-for-Testability and Built-In-Self-Test (BIST) schemes. This paper presents the state of art in memory testing including fault modeling, test design and BIST. Further research challenges and opportunities are discussed in enabling testing (embedded) memories in the nano-era.

Keywords: Memory Test, Fault Modeling, test algorithm design, MBIST.

1 Introduction

The semiconductor memory development over the years can be classified in three phases [1]: (a) the stand-alone, (b) memory integrated with logic and (c) scaled embedded memory. In the first phase, typically from about 1980 to 1990, the ideal MOS memory was a standardized stand-alone part, with its small cell size, good array efficiency, adequate performance, noise and soft error resistance, and met an external I/O standard. In the second phase, occurred from 1990 to 2000, where memories began to be integrated onto the logic chip; however, embedded DRAM and Flash were hindered by the historical divergence of the memory and logic technologies. In the third phase, from 2000 on, the era of true embedded memory has begun. Nowadays, embedded memories represent the great majority of embedded electronics in Systems on Chip (SoC). It is very common to find SoCs with hundreds of memories representing more than 50% of the overall chip area. According to the ITRS, today's SoCs have been moving from logic-dominant to memory-dominant chips in order to deal with the requirements of todays and future applications. Consequently, embedded memory test challenges will significantly impact the overall testability of SoC. Solving such challenges for memories will substantially contribute to the resolution of electronic system test problems in the future; hence, supporting the continuation of the semiconductor technology revolution and the manufacturability of future highly complex systems ('gigascale') and highly integrated technologies ('nano-scale').

The cost of memory testing increases with every generation of new memory chips [2]. Precise *fault modeling* to design *efficient tests*, in order to keep the test cost and test time within economically acceptable limits, is therefore essential. The quality of the tests, in terms of defect/fault coverage, is strongly dependent on the used fault models. Therefore, fault models reflecting the real defects of the new memory technologies are crucial for developing higher quality test algorithms and therefore providing products with low Defect-Per-Million (DPM) level driven by the market.

It is difficult, if not impossible, to test an embedded memory simply by applying test patters directly to the chip's I/O pins, because the embedded memory's address, data, and control signals are usually not directly accessible through the I/O pins. Therefore, *Memory Built-In-Self Test (MBIST)* is used for memory testing. The basic philosophy behind the MBIST technique is: "let the hardware test itself "; i.e. enhance the functionality of the memory to facilitate self-test. Large (and expensive) external tests cannot provide the needed test stimuli to enable high speed, nor high quality tests [3]. BIST therefore is the only practical and cost-effective solution for embedded SoC memories.

This paper addresses the state-of the art of the three major aspects related to embedded memory testing; these are fault modeling, test algorithm design and MBIST. In addition, future challenges and trends will be covered.

2 Memory Fault Modeling

As already mentioned, the quality of fault models used to develop test algorithms is very crucial in providing high outgoing product quality measured in DPM level. The concept of memory fault model appeared first in early 1980's. Many fault models have been developed targeting different fault behaviors. Memory fault models can be classified into two classes: (a) *static faults*, developed mainly between 1980 and 2000, and (d) *dynamic faults*, have been developed since begin 2000.

Static Faults

During the early 1980's many memory functional fault models have been introduced, allowing the fault coverage of a certain test to be provable. Some important fault models introduced in that time are [4,5,6]: Stuck-at-Faults and Address-Decoder-Faults. These are abstract fault models and are not based on any real memory design and/or real defects. To reflect the faulty behavior of the real defects in real designs, Inductive Fault Analysis (IFA) was introduced. IFA allows for the establishment of the fault models based on simulated defects at the physical layout level of the design. In addition, IFA is capable of determining the occurrence probability and the importance of each fault model. The result was that new fault models were introduced [7]: State-Coupling Fault and Data-Retention Fault. In the late 1990's, experimental results based on DPM screening of a large number of tests applied to a large number of memory chips indicated that many detected faults cannot be explained with the well-known fault models [8,9], which suggested the existence of additional faults. This stimulated the introduction of new fault models, based on defect injection and SPICE simulation [10,11]: Read Destructive Fault, Write Disturb Fault, Transition Coupling Fault, Read Destructive Coupling Fault, etc.

All the memory fault modeling described above focuses on faults sensitized by performing *at most one operation*. For instance, Read Destructive Coupling Fault is sensitized by applying a read operation to the victim cell, while the aggressor cell is put in a certain state (i.e., the required number of operations is 1). Memory faults sensitized by performing at most one operation are referred to as *static faults*.

Dynamic Faults

Many work published since early 2000 have revealed the existence and the importance of another class of faults in the new memory technologies. It was shown that another kind of faulty behavior can take place in the absence of static faults [12]-[21]. This faulty behavior has been attributed to *dynamic faults*, which require *more than one operation to be performed sequentially* in time in order to be sensitized. For example, a write 1 operation followed immediately by a read 1 operation will cause the cell to flip to 0; however, if only a single write 1 or a single read 1, or a read 1 which is not immediately applied after write 1 operation is performed, then the cell will not flip. [12] observed the existence of dynamic faults in the new embedded DRAMs based on defect injection and SPICE simulation. [13,14] observed the presence of dynamic faults in embedded caches of Pentium processors during a detailed analysis of the DPM screening results of a large number of tests. [15,16] showed the importance of dynamic faults for new SRAM technologies by analyzing DPM screening results of Intel and STMicroelectronics products, and concluded that current and future SRAMs tests need to consider dynamic faults or leave substantial DPM on the table. [17] showed the existence of dynamic memory cell array faults for SRAMs using defect injection and circuit simulation. The work of [18,19] proved the existing of dynamic faults in the peripheral circuits of a memory (such as sense amplifiers, pre-charge circuits, etc.), while the work of [20,21] showed the existence of this class of faults in the address decoders. Due to the importance of covering dynamic faults in order to realize the required product quality (as it has been show by measured data [14]-[16], [22]), tests for such faults become an integral part of memory test programs used within industry [23,24,25]. Dynamic faults are becoming even more important for the future technologies [26].

3 Test Algorithm Design

Tests and fault detection for semiconductor memories have experienced a long evolutionary process, starting before 1980's. Overall, memory test algorithms can be classified into three classes: (a) Ad-hoc tests, (b) March tests, and (b) Fault primitive based tests; they are explained next.

Ad-hoc Tests: The are the early tests (typically before the 1980's). They are classified as Ad-Hoc because of the absence of formal fault models and proofs. Tests as Scan, Checkerboard, Galpat and Walking 1/0 [5,27] belong to this class. They have further the property of either having a low fault coverage (as it is the case for Scan and Checkerboard) or requiring a very long test time (as it is the case for Galpat and Walking), which make them very uneconomical for larger memories.

March Tests: After the introduction of fault models during the early of 1980's, march tests became the dominant type of tests. The advantages of march tests lay in two facts. First, the fault coverage of the considered/known models could be mathematically proven, although one could not have any idea about the correlation between the models and the defects in the real chips. Second, the test time for march tests is typically linear with the size of the memory, which made them acceptable from an industrial point of view. Some well-known march tests, that have been shown to be efficient, are: Mats+ [28], March C- [29], PMOVI [30], IFA 13n [7], etc.

Fault Primitive Based Tests: As new fault models have been introduced in the late 1990's, based on defect injection and SPICE simulation, other new tests have been developed to deal with them. In addition, the concept of Fault Primitive (FP) was introduced to better describe a *single* fault behavior [31]; it made it easier to analyze the failure mechanisms in deep-sub micron technology and to develop realistic fault models and thereafter optimal test algorithms using defect-oriented testing. This approach allows for the realization of high defect coverage as the algorithm can be optimized for each design/layout. Some of the introduced FP based tests are targeting static faults; examples are March SS [32] and March MSS[33] for detecting all possible static faults, March SR [11] for detecting faults realistic for the design under consideration, etc. However, most of the developed FP based tests are targeting dynamic faults, including memory cell array faults [13]-[16], peripheral circuits faults [18,19] and address decoder faults [20,21,34].

It is worth noting that the state-of-the art in memory fault modeling and test design typically assumes the presence of a single fault at a time; memory tests assuming the presence of multiple faults at a time have got limited attention in the past [35,36] and they seems to become more important with further technology scaling [37]. Fault modeling and test design have to consider not only the presence of a single defect/fault at a time, but also the presence of multiple weak-faults/defects simultaneously (this is particularly important in the nano-era). A weak fault is not able to fully sensitize a fault, but it partially sensitizes it; e.g., due to a partial open defect that creates a small delay. However, a fault can be fully sensitized (i.e., becomes strong) when two (or more) weak faults are sensitized *simultaneously* since their fault effects can be additive.

4 Memory Built-In-Self Test

As already mentioned, it is difficult to test an embedded memory simply by applying test patters directly to the chip's I/O pins, because the embedded memory's address, data, and control signals are usually not directly accessible through the I/O pins. Therefore, BIST is the only practical and cost-effective solution for embedded SoC memories.

BIST engines, no matter what kind, can use pseudo-random or deterministic test patterns/algorithm. A pseudo-random pattern is basically very helpful to test logic circuits. A memory, however, has a regular structure and typically requires the application of regular and deterministic test patterns as those discussed in the previous section. In the early days of BIST (typically before 1990's), it was not unusual to see pseudo-random techniques applied to memory [38,39]. However, this approach has been hardly used from 1990 on due to its low fault coverage.

MBIST based on deterministic patterns is dominant for testing memories today. Deterministic patterns means that the patters are generated according to specified predetermined values (such as march tests). When implementing MBIST engines, trade-offs are made depending on: (a) the number of supported algorithms, (b) the flexibility of the MBIST engine (in order to cope with the unexpected), (c) the implementation speed, and (d) the area overhead. In addition, when MBIST performs tests, memory accesses have to be done at-speed using Back-to-Back memory cycles in order to detect dynamic faults [14]-[25]. Systems require large, high speed memories, while the technology scaling exhibits a large spread in implementation parameters, resulting in speed related (dynamic) faults.

Many papers have been published on deterministic based MBIST; examples are [40]- [50]. Let's assume that every memory test algorithm can be described using an extended notation of March algorithms [27], including the non-linear algorithms such as Galpat and Walking 1/0. A march test consists of a finite sequence of March Elements (MEs) [4]; a march element is a finite sequence of operations applied to every cell in the memory before proceeding to the next cell. Based on the level at which memory algorithms are specified, MBIST engines can be classified into four classes.

Algorithm Based MBST: Only a single (or few) algorithms can be specified; they are implemented in hardware using a state machine [40,41]. This MBIST is generally used in industry to generate a single pattern (e.g., a single march test). However, a better memory test solution requires a suite of patterns; this makes the design of the state machine complex. The major limitation of algorithm based MBIST lays in its quite restricted flexibility; modifying the patterns requires changing the MBIST design. This MBIST implementation was used in the early days of MBIST, where the fault behavior was still considered simple and mainly static, while the implementation cost was very important. Performing at speed-testing was not a crucial issue for such MBIST class.

March Element Based MBIST: Applying an algorithm consists of successively scanning-in each of the algorithm's MEs, together with its algorithm stress combination. In order to be able to perform memory tests at-speed, it is sufficient that each of the individual MEs is performed at-speed [42,45,49,50]. For this MBIST class, not the algorithms are hard-wired, but only the MEs. Recent publications [49,50] show that such MBIST class can provide higher flexibility at extremely small command memory, which makes this MBIST class very attractive for embedded applications whereby the tests have to be stored within the MBIST engine. Moreover, all information required to support an at-speed implementation is contained in the specified MEs. Hence, the required detailed information to control the memory, such as whether a read or write has to be performed, can be decoded from the ME prior to its application. This prevent the complex implementation (costly hardware) which typically uses complex schemes such as pipelining and prefetching to apply at speed-testing.

ME operations Based MBIST: In this case, at speed of only the operations within a single ME can be performed; elapsed time between MEs is not critical. Typically each ME can use a certain stress combination (such as data-background). Therefore this class can be easy implemented using two registers (one for ME and one for the stress combination) and a register controlled state-machine; the two register can be scanned via a low speed tester. Because of its low cost implementation, such MBIST is very popular within the industry. Note that ME operations based MBIST requires more test time than March Element Based MBIST; in the latter all MEs are hard-wired and no external scanning is needed for MEs to perform the tests.

Individual Operations Based MBIST: This class of MBIST engines allows for the specification of algorithms by specifying each of the algorithm operations. Obviously, this can be very flexible. However, the implementation cost is effectively determined by the size of the algorithm memory and the supported addressing scheme(s) [46]-[48], [51]. The more algorithms and addressing schemes, the more hardware overhead. In addition, it typically explores expensive prefetching and pipelining techniques to perform at speed-testing, which makes the implementation costly. This class is mainly suitable for higher end products.

5 Future Challenges

This section addresses some major challenges and trends wrt embedded memory testing and gives some research directions. First, the technology technology threats due to continue scaling will be discussed. Thereafter, the business pressure will be covered. Finally, the requirements for future memory test solutions will be described.

5.1 Technology Threats

Progressive technology scaling, as tracked by the ITRS and encapsulated by Moore's law, has driven the phenomenal success of the semiconductor industry. Silicon technology has now entered the nano-era and the 10nm transistors are expected to be in production by 2018, allowing the integration of a wider variety of functions. However, it is well recognized that many challenges are emerging:

– *Extreme variations*: The increasing variability in device characteristics and its impact on the overall quality and reliability represent major challenges to scaling and integration for future nanotechnology generations [26] (cross talk, interferences, leakages, Vth mismatch, degraded Read/Write margins, ...). What is more, newly emerging complex failure mechanisms in the nano-era (which are not understood yet), are causing the fault mode of the chips to be dominated by transient, intermittent, parametric and weak faults rather than hard and permanent faults; hence causing more reliability problems than quality problems [52]-[54] .
– *Reduced voltages*: Although the supply voltage is not scaling at the same scale as the technology, the reduced voltages are contributing to the emergence of many new failure mechanisms that may impact either the quality and/or the reliability of memories; examples are: reduced signal strength, reduced SNM (higher soft error rates) and increased sensitivity to delay (parametric) faults.
– *Speed related faults*: The increasing clk speed of each memory generation poses new challenges. For instance, a 10ps delay for a memory running at 100MHz is only 1% of the Clk speed. However, this is 10% for a memory running at 1GHz! What is used to be known as marginal delays can cause timing failures for today and future technologies. Hence testing for at-speed related faults is becoming a must. Not to mention that other emerging failure mechanisms also contribute to speed related faults.
– *Wearout*: The aggressive technology scaling does not only cause new failure mechanisms, but also increases the level of transient errors (during device lifetime) and reduces the device lifetime, causing major reliability challenges.

5.2 Business Pressure

The continuous increases in SoC complexity in general and the simultaneous higher competitive semiconductor industry pose many business pressure challenges.

– *Higher customer requirements*: irrespective of the business pressure, customers always require a higher product quality, lower cost and higher reliable chips. Higher customer satisfaction requires therefore *higher fault coverage*, even for unknown faults. Hence, large set of tests and stresses are needed, making test cost higher (reducing benefits).

- *Shorter time-to-volume (TTV)/market*: TTV consist of two parts: (a) design time and (b) production ramp-up time [1]. IP reuse is a common practice used to reduce the design time. Reducing production ramp-up time is what is causing the real bottleneck. Traditionally, one had sufficient learning time for test cost and DPM level reduction, starting at low volume with a large number of tests used to detect, analyze and correct yield problems. Today, and due to time-to-market pressure, the time-to-market is reduced causing a severe reduction in the learning curve. Due to the very short to no learning curve, understanding of all faults for each new technology is impossible. Hence, the test program may provide a *low fault coverage*, which contradicts the customer requirements.

5.3 Requirements for Future Test Solutions

Given the challenges mentioned above, the future solutions have to provide answers to both the customer requirements (higher quality, reliability and fault coverage for all emerging faults) and the short time-to-market (higher yield). Today's solutions are going in the following direction:

- *Manufacturing Test*: Use effective test set. Development of new tests may require new approaches. To optimize cost v quality (wrt time-to-volume), this test may not detect all faults (economically undetectable faults escape). In addition, use a programmable BIST at module level to enhance shorter time to market
- *In field Test*: Use Error Correction codes (ECC) to compensate for incomplete fault coverage and to detect soft errors and new (unexpected) failures. In addition, use dynamic Built-In-Self-Repair to maintain the effectives of ECC and increase yield and product lifetime.

6 Conclusion

This paper has discussed three major aspects related to embedded memory testing and has provided some future challenges. The approach based on single defect a time causing a *strong fault*, on which the traditional fault modeling and test design are based, may need refinement as memories in nano-era may suffer from different small disturbances (*weak faults*) at the same time; these weak faults can together create a strong fault if sensitize *simultaneously* during the application. Therefore a new memory test paradigm may be needed.

References

1. Marinissen, E.J., et al.: Challenges in Embedded Memory Design and Test. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (2005)
2. Inoue, M., et al.: A New Test Evaluation Chip for Lower Cost Memory Tests. IEEE Design and Test of Computers 10(1), 15–19 (1993)

3. Mookerjee, R.: Segmentation: A Technique for Adapting High-Performance Logic ATE to Test High-Density, High-Speed SRAMs. In: IEEE Workshop on Memory Test, pp. 120–124 (1993)
4. Suk, D.S., Reddy, S.M.: A March Test for Functional Faults in Semiconductors Random-Access Memories. IEEE Transactions on Computers C-30(12), 982–985 (1981)
5. Abadir, M.S., Reghbati, J.K.: Functional Testing of Semiconductor Random Access Memories. ACM Computer Surveys 15(3), 175–198 (1983)
6. Papachistou, C.A., Saghal, N.B.: An Improved Method for Detecting Functional Faults in Random-Access Memories. IEEE Trans. on Computers C-34(2), 110–116 (1985)
7. Dekker, R., et al.: A Realistic Fault Model and Test Algorithms for Static Random Access Memories. IEEE Trans. on Computers 9(6), 567–572 (1990)
8. Schanstra, I., van de Goor, A.J.: Industrial evaluation of Stress Combinations for March Tests Applied to SRAMs. In: Proc. IEEE Int. Test Conference, pp. 983–992 (1999)
9. van de Goor, A.J., de Neef, J.: Industrial Evaluation of DRAMs Tests. In: Proc. of Design Automation and Test in Europe, pp. 623–630 (1999)
10. Adams, D., Cooley, E.S.: Analysis of Deceptive Read Destructive Memory Fault Model and Recommended Testing. In: Proc. of IEEE NATW (1999)
11. Hamdioui, S., van de Goor, A.J.: Experimental Analysis of Spot Defects in SRAMs: Realistic Fault Models and Tests. In: Proc. of Ninth Asian Test Symposium, pp. 131–138 (2000)
12. Al-Ars, Z., van de Goor, A.J.: Static and Dynamic Behavior of Memory Cell Array Opens and Shorts in Embedded DRAMs. In: Proc. of Design Automation and Test in Europe, pp. 401–406 (2001)
13. Hamdioui, S., Al-ars, Z., van de Goor, A.J.: Testing Static and Dynamic Faults in Random Access Memories. In: Proc. of IEEE VLSI Test Symposium, pp. 395–400 (2002)
14. Hamdioui, S., van de Goor, A.J., Reyes, J.R., Rodgers, M.: Memory Test Experiment: Industrial Results and Data. IEE Proceedings of Computers and Digital Techniques 153(1), 1–8 (2006)
15. Hamdioui, S., et al.: Importance of Dynamic Faults for New SRAM Technologies. In: Proc. of European Test Workshop, pp. 29–34 (2003)
16. Hamdioui, S., Wadsworth, R., Reyes, J.D., Van De Goor, A.J.: Memory Fault Modeling Trends: A Case Study. Journal of Electronic Testing 20(3), 245–255 (2004)
17. Dilillo, L., et al.: Dynamic read destructive fault in embedded-SRAMs: analysis and march test solution. In: Proc. Ninth IEEE European Test Symposium, pp. 140–145 (2004)
18. Van de Goor, A.J., Hamdioui, S., Wadsworth, R.: Detecting faults in the peripheral circuits and an evaluation of SRAM tests. In: Proc. of Inter. Test Conference, pp. 114–123 (2004)
19. Dilillo, L., et al.: Resistive-Open Defect Influence in SRAM Pre-Charge Circuit: Characterization and Analysis. In: 10th European Test Symposium on IEEE ETS 2005 (2005)
20. Hamdioui, S., Al-Ars, Z., van de Goor, A.J.: Opens and Delay Faults in CMOS RAM Address Decoders. IEEE Trans. on Computers 55(11), 1630–1639 (2006)
21. Dilillo, L., et al.: ADOFs and Resistive-ADOFs in SRAM Address Decoders: Test Conditions and March Solutions. Jour of Electronic Testing: Theory and Applications 22(3), 287–296 (2006)

22. Hamdioui, S., Al-Ars, Z., Jimenez, J., Calero, J.: PPM Reduction on Embedded Memories in System on Chip. In: IEEE Proc. of European Test Symposium, Freiburg, Germany, pp. 85–90 (May 2007)
23. Al-Ars, Z., Hamdioui, S., Gaydadjiev, G.N., Vassiliadis, S.: Test Set Development for Cache Memory in Modern Microprocessors. IEEE Trans. Very Large Scale Integration (VLSI) Systems 16(6), 725–732 (2008)
24. Powell, T., Kumar, A., Rayhawk, J., Mukherjee, N.: Chasing Subtle Embedded RAM Defects for Nanometer Technologies. In: Proc. of the IEEE Int. Test Conf., paper 33.4, vol. 23(5) (October 2007)
25. Mukherjee, N., Pogiel, A., Rajski, J., Tyszer, J.: High Volume Diagnosis in Memory BIST Based on Compressed Failure Data. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 29, 441–453 (2010)
26. Bhavnagarwala, A., et al.: The semiconductor industry in 2025. In: IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 534–535 (2010)
27. van de Goor, A.J.: Testing Semiconductor Memories, Theory and Practice, 2nd edn. ComTex Publishing, Gouda (1998)
28. Nair, R.: An Optimal Algorithm for Testing Stuck-at Faults Random Access Memories. IEEE Trans. on Computers C-28(3), 258–261 (1979)
29. Marinescu, M.: Simple and Efficient Algorithms for Functional RAM Testing. In: Proc. of IEEE International Test Conference, pp. 236–239 (1982)
30. De Jonge, J.H., Smeulders, A.J.: Moving Inversions Test Pattern is Thorough, Yet Speedy. In: Comp. Design, pp. 169–173 (1979)
31. van de Goor, A.J., Al-Ars, Z.: Functional fault models: A formal notation and taxonomy. In: Proc. IEEE VLSI Test Symp., pp. 281–289 (2000)
32. Hamdioui, S., van de Goor, A.J., Rodgers, M.: March SS: A Test for All Static Simple RAM Faults. In: Proc. of IEEE International Workshop on Memory Technology, Design, and Testing, Bendor, France, pp. 95–100 (2002)
33. Harutunvan, G., Vardanian, V.A., Zorian, Y.: Minimal March tests for unlinked static faults in random. In: Proc. of IEEE VLSI Test Symposium, pp. 53–59 (2005)
34. van de Goor, A.J., Hamdioui, S., Gaydadjiev, G.N., Al-Ars, Z.: New Algorithms for Address Decoder Delay Faults and Bit Line Imbalance Faults. In: 18th IEEE Asian Test Symposium, Taichung, Taiwan, pp. 391–397 (November 2009)
35. van de Goor, A.J., et al.: March LA: A test for linked memory faults. In: Eur. Design Test Conf., p. 627 (1999)
36. Hamdioui, S., et al.: Linked faults in random access memories: concept, fault models, test algorithms, and industrial results. IEEE Trans. on CAD of Integrated Circuits and Systems 23(5), 737–757 (2004)
37. Hamdioui, S., et al.: A New Test Paradigm for Semiconductor Memories in the Nano-Era. In: Proc. of Asian Test Symposium, pp. 347–352 (2011)
38. McAnney, et al.: Random Testing for Stuck-At Storage Cells in an Embedded Memory. In: Proc. of Intern. Test Conference, pp. 157–166 (1984)
39. David, R., Fuentes, A., Courtois, B.: Random Patterns Testing Versus Deterministic Testing of RAMs. IEEE Trans. on Computers 5, 637–650 (1989)
40. Dekker, R., Beenker, F., Thijssen, L.: Realistic built-in self-test for static RAMs. Design & Test of Computers 6(1), 26–34 (1989)
41. Dreibelbis, J.H., Hedberg, E.L., Petrovic, J.G.: Built-In Self-Test for Integrated Circuits, US Patent, Number 5,173,906 (December 22, 1992)
42. Zarrineh, K., et al.: A new framework for generationg optimal March tests for memeory arrays. In: Proc. of the Int. Test Conf., pp. 73–82 (1998, 2001)

43. Powell, T.J., et al.: BIST for Deep Submicron ASIC Memories with High Performance Application. In: Proc. of the IEEE Int. Test Conf., pp. 386–392 (2003)
44. Appello, D., et al.: Exploiting Programmable BIST For The Diagnosis of Embedded Memory Cores. In: Int. Test Conference, p. 379 (2003)
45. Aitken, R.C.: A Modular Wrapper Enabling High Speed BIST and Repair for Small Wide Memories. In: Proc. of the IEEE Int. Test Conf., paper 35.2, pp. 997–1005 (2004)
46. Du, X., Mukherjee, N., Cheng, T.M.: Full-Speed Field-Programmable Memory BIST Architecture. In: Proc. of the IEEE Int. Test Conf., paper 45.3 (2005)
47. Du, X., Mukherjee, N., Cheng, W.-T., Reddy, S.M.: A Field-Programmable Memory BIST Architecture Supporting Algorithms and Multiple Nested Loops. In: Proc. of the Asian Test Symposium, paper 45.3 (2006)
48. van de Goor, A.J., Jung, C., Gaydadjiev, G.: Low-cost, Flexible SRAM MBIST Engine. In: IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (April 2010)
49. van de Goor, A.J., Hamdioui, S., Gaydadjiev, G., Alars, Z.: Generic March Element Based Memory Built-In Self-Test, Dutch Patent Application; Filing Number NL 2004407, Filed date (January 2010)
50. van de Goor, A.J., Hamdioui, S., Kukner, H.: Generic, orthogonal and low-cost March Element based memory BIST. In: Inter. Test Conference, pp. 1–10 (2011)
51. Khare, J.B., Shah, A.B., Raman, A., Rayas, G.: Embedded Memory Field returns - Trials and Tribulations. In: Proc. IEE Int. Test Conf., Paper 26.3 (2006)
52. Borkar, S.: Design and Test Challenges for 32 nm and Beyond. Keynote speech at IEEE International Test Conference, p. 13 (2009)
53. Hamdioui, S., Al-Ars, Z., Mhamdi, L., Gaydadjiev, G.N., Vassiliadis, S.: Trends in Tests and Failure Mechanisms in Deep Sub-micron Technologies. In: IEEE Proc. of Int. Conference on Design and Test of Integrated Systems in Nanoscale Technology, pp. 216–221 (September 2006)
54. Vermeulen, B., Hora, C., Kruseman, B., Marinissen, E.J., van Rijsinge, R.: Trends in Testing Integrated Circuits. In: Proc. IEEE Int'l Test Conf., pp. 688–697 (2004)

Quicksort and Large Deviations

Colin McDiarmid

University of Oxford

Abstract. Quicksort may be the most familiar and important randomised algorithm studied in computer science. It is well known that the expected number of comparisons on any input of n distinct keys is $\Theta(n \ln n)$, and the probability of a large deviation above the expected value is very small. This probability was well estimated some time ago, with an ad-hoc proof: we shall revisit this result in the light of further work on concentration.

1 Introduction

Quicksort was introduced in the 1960's by Hoare [7,8], and has established itself as one of the classic algorithms of computer science. There are several reasons for this: it is efficient and useful in practice; it illustrates two key ideas in algorithm design, namely divide and conquer, and randomization; and the study of quicksort and its variants has become a model for the analysis of algorithms in general.

Let us focus on the number of comparisons required for sorting, as is usual. As we shall see below, the expected number of comparisons is suitably small. The aim of this paper is to revisit the rather precise estimation in Hayward and McDiarmid [13,14] on

the probability that the number of comparisons of a random execution of quicksort will have a large deviation from the expected number of comparisons.

The original proof for the upper bound on the probability involved adapting, in a rather ad-hoc way, parts of the proofs from the then recently popularised combinatorial approach known as the "method of bounded differences" for establishing concentration [11]. We shall rework the proof, in the light of further general work on concentration.

Before introducing the definitions we need in order to state our theorems precisely, let us discuss the two variants of quicksort we shall refer to in this paper. First, by "basic" quicksort (and unless otherwise stated, that is the version we will be referring to) we mean the original, unadorned version:

A partitioning key is selected uniformly at random from the list of unsorted keys, and used to partition the keys into two sublists. The algorithm is called recursively on remaining unsorted sublists, until sublists have size one or zero.

One common variant is to use a different sorting algorithm (usually insertion sort) for lists whose size is not greater than a certain threshold value M. We refer to this variant as "cutting at length M". Another common variant is to select the partitioning key as the median of $2t+1$ keys selected uniformly at random from the list of unsorted keys. For an introduction to the study of quicksort see for example Cormen, Leiserson and Rivest [1]: for a comprehensive survey and comparative analysis of common variants see Sedgewick [17]. For some more recent relevant results on the analysis of quicksort, see [6], [10], [15], [16].

We now introduce some notation. Let Q_n be the random number of key comparisons made when (basic) quicksort sorts n keys. We make the usual assumption that the n keys are distinct. (Alternatively, we could assume that all $n!$ linear orders are equally likely, and then our results apply to a suitable deterministic version of quicksort.) A straightforward and well known analysis shows that the expected number $q_n = \mathbb{E}[Q_n]$ of key comparisons satisfies $q_0 = 0$ and for $n \geq 1$

$$q_n \;=\; n-1 \;+\; \frac{1}{n}\sum_{j=1}^{n}(q_{j-1}+q_{n-j})\,. \tag{1}$$

The harmonic numbers H_n are defined by $H_0 = 0$ and $H_t = \sum_{j=1}^{t} 1/j$ for all integers $t \geq 1$. It follows easily from Equation (1) that $q_n = 2(n+1)H_n - 4n$. Since $H_n = \ln n + \gamma + O(1/n)$ as $n \to \infty$ (where $\gamma \approx 0.577$ is Euler's constant) it follows that

$$q_n \;=\; 2n\ln n \;-\; (4-2\gamma)n \;+\; 2\ln n \;+\; O(1)\,.$$

Of course one attaches more credibility to an average case result if it is known that there is a strong concentration of probability around the mean. In particular, given $\epsilon > 0$, what bounds can be placed on the quantity

$$\mathbb{P}[\;|\frac{Q_n}{q_n} - 1| > \epsilon\,] \;\;?$$

We want to show tight bounds for the above probability, as in [13,14], but with a less ad-hoc proof for the upper bound. There were previous bounds known. For instance, from the fact that the variance of Q_n is $\Theta(n^2)$ (see [9] or [17]), it follows using Chebyshev's inequality that for each $\epsilon > 0$

$$\mathbb{P}[\;|\frac{Q_n}{q_n} - 1| > \epsilon\,] \;=\; O((\epsilon \ln n)^{-2})\,.$$

Hennequin [6] used Chebyshev's inequality with fourth moments to show that for each $\epsilon > 0$ the above probability is $O((\epsilon \ln n)^{-4})$. Then Rösler [16] much improved these upper bounds, and showed that for each fixed $\epsilon > 0$ this probability is $O(n^{-k})$ for any fixed k.

The probability of such deviations is even smaller than this. For each fixed $\epsilon > 0$ we shall see that

$$\mathbb{P}[\;|\frac{Q_n}{q_n} - 1|\; > \epsilon] \;=\; n^{-(2+o(1))\,\epsilon \ln^{(2)} n} \tag{2}$$

as $n \to \infty$. Note that here we use $\ln n$ to denote $\log_e n$. Also, we use the notation $\ln^{(k)} n$, where $\ln^{(1)} n = \ln n$ and $\ln^{(k+1)} n = \ln(\ln^{(k)} n)$. The main result here for basic quicksort is a more precise version of the equation (2).

Theorem 1. (Hayward and McDiarmid [13,14])
Let $\epsilon = \epsilon(n)$ satisfy $1/\ln n < \epsilon \leq 1$. Then as $n \to \infty$,

$$\mathbb{P}[\, |\frac{Q_n}{q_n} - 1| > \epsilon] \;=\; n^{-2\epsilon(\ln^{(2)} n - \ln(1/\epsilon) + O(\ln^{(3)} n))}\,.$$

This result is quite precise for $\epsilon > 0$ fixed or tending to 0 very slowly, though it says little when $\epsilon = O(\ln^{(2)} n/\ln n)$.

Theorem 1 is stated above for the most basic form of quicksort. However, the difference between the number of comparisons required by this version of quicksort and the corresponding number for the more efficient "cutting at length M" variant is only $O(n)$. This small term does not affect the result, and the theorem just stated (as well as the one that is to follow) also holds for the "cutting at length M" variant.

Now consider the 'median of $(2t+1)$' variants of quicksort. Recall that these are the variants in which the partitioning key is chosen as the median of $(2t+1)$ keys. Here t is a fixed non-negative integer, $t = 0$ corresponds to basic quicksort, and $t = 1$ is perhaps most common in practice.

We need more notation before stating the second and final theorem. Let $Q_n^{(t)}$ be the random number of comparisons taken to sort a list of length n, and let $q_n^{(t)} = \mathbb{E}[Q_n^{(t)}]$. Thus $Q_n^{(0)}$ is Q_n. For $j = 1, 2, \ldots$ let $\kappa_j = (H_{2j+2} - H_{j+1})^{-1}$. Thus for example $\kappa_0 = 2$ and $\kappa_1 = 12/7$. It is well known ([19] and [17]) that

$$q_n^{(t)} \;=\; \kappa_t\, n \ln n \;+\; O(n) \qquad \text{as} \;\; n \to \infty\,.$$

We can now state the 'median of $(2t+1)$' extension of the first theorem:

Theorem 2. (Hayward and McDiarmid [13,14])
Let $\epsilon = \epsilon(n)$ satisfy $1/\ln n < \epsilon \leq 1$. Then as $n \to \infty$,

$$\mathbb{P}[\,|\frac{Q_n^{(t)}}{q_n^{(t)}} - 1| > \epsilon] = n^{-(t+1)\kappa_t\epsilon(\ln^{(2)} n - \ln(1/\epsilon) + O(\ln^{(3)} n))}\,.$$

In the next section we give some fundamental lemmas about basic quicksort, and after that we introduce the concentration result, Lemma 6, which we shall use in the proof of the upper bound in Theorem 1. A similar proof using Lemma 6 will yield the upper bound in Theorem 2 for median of $(2t+1)$ quicksort, though the 'fundamental lemmas' are more involved, see [14]: we will not pursue this here.

Recall that the results for basic quicksort apply to the analysis of binary search trees, via a well known correspondence which we now describe. An execution of quicksort may be associated with a 'partition tree', namely the tree whose root is the initial partitioning key, and whose two subtrees are the trees of the two recursive calls. Given a fixed number of keys, there is a one-to-one correspondence between partition trees and binary search trees (binary trees in which each left/right child is lesser/greater than its parent). Thus the distribution of the number of key comparisons of a quicksort execution is the same as the distribution of the number of key comparisons of a binary search tree construction, assuming that binary search trees are constructed by repeated equi-probable insertion of keys not yet in the tree. Thus Theorem 1 can be interpreted as a result on binary search trees, by letting Q_n be the number of key comparisons made when a binary search tree on n keys is constructed.

We close this section by giving a quick overview on large deviations for basic quicksort and how the upper and lower bound proofs work. In this paper we focus only on the upper bound, but it helps also to see how the lower bound proof works. We associate with an execution of quicksort a binary tree like the partition tree above, where the root contains the original unsorted list, and the children of any node contain the sublists obtained by splitting the list at the parent node.

Firstly, when will we get a large deviation? For a suitable integer k (of order $\ln n$) consider the first k nodes on the leftmost path from the root in the binary tree, and consider the set of their right children. For a suitable integer ℓ (of order a little less than $n/\ln n$) let A be the event that the list length at each of these right children is at most ℓ (so we have a succession of k bad splits). Then $\mathbb{P}[A]$ is as on the right side of the equation in Theorem 1. Using the fact that the variance of Q_n is $O(n^2)$ we may see that $\mathbb{P}[Q_n > (1+\epsilon)q_n | A] = 1 + o(1)$; and thus

$$\mathbb{P}[Q_n > (1+\epsilon)q_n] \geq \mathbb{P}[Q_n > (1+\epsilon)q_n | A]\,\mathbb{P}[A] = (1+o(1))\,\mathbb{P}[A].$$

How do we obtain the upper bound? As we move from the root through the binary tree, think of step j as choosing the partitioning key for the list at node j (and performing the relevant comparisons). Let X_j be the conditional expectation of Q_n given what we have seen up to step j. We will see that $|X_j - X_{j-1}|$ is at most the length of the list at node j. Thus the sum of these values over all the nodes j at a given depth is at most n; and once we are well down the tree (depth of order $\ln n$) with very high probability the list lengths are all small, and then the conditional range of X_j is small. This allows us to use a martingale concentration inequality to obtain the upper bound as required.

2 Basic Quicksort

In this section we present the fundamental lemmas about basic quicksort which we use in the proof of the upper bound in Theorem 1, with sketch proofs. For full proofs see [14]. Let us first describe more precisely the correspondence mentioned above between an execution of quicksort and an appropriate binary tree.

Consider the infinite binary tree, with nodes numbered $1, 2, 3, \ldots$ level by level and left to right in the usual manner. (So for instance, the path from the root down the left side is $1, 2, 4, 8, \ldots$). Each execution of (basic) quicksort yields a labelling of a subtree of this tree, corresponding to the recursive structure of quicksort. The root, node 1, is labelled with the unsorted list of n keys, and its 'list length' L_1 is n. A partitioning key is chosen, and after partitioning an (unsorted) list of those keys less than the partitioning key forms the label for the left child (node 2) which then acts like the root of a new tree. Similarly, those keys greater than the partitioning key are sent to the right child of the root.

For each $j = 1, 2, \ldots$ let L_j be the length of the list to be sorted at node j. Thus $L_1 = n$ and only n of the L_j are non-zero. The first lemma below shows that as we move down the tree, the list lengths shorten suitably with high probability. Let M_k^n be the maximum value of the list length L_j over the 2^k nodes j at depth k, that is

$$M_k^n \ = \ \max\{L_{2^k+i} : i = 0, 1, \ldots, 2^k - 1\} \, .$$

Lemma 1. *For any* $0 < \alpha < 1$ *and any integer* $k \geq \ln(1/\alpha)$

$$\mathbb{P}[M_k^n \geq \alpha n] \ \leq \ \alpha \left(\frac{2e \ln(1/\alpha)}{k} \right)^k \, .$$

Proof. (*Sketch.*) The key observation is that we can obtain the exact joint distribution of $(L_1, L_2, \ldots)$ as follows. Here we are developing an idea of Devroye [2].

Let the random variables $X_1, X_2, \ldots$ be independent with each uniformly distributed on the interval (0,1). Define random variables $\tilde{L}_1, \tilde{L}_2, \ldots$ as follows. Let $\tilde{L}_1 = n$ and for $i \geq 1$ let $\tilde{L}_{2i} = \lfloor X_i \tilde{L}_i \rfloor$ and $\tilde{L}_{2i+1} = \lfloor (1 - X_i) \tilde{L}_i \rfloor$. Then it is easily seen that $(L_1, L_2, \ldots)$ and $(\tilde{L}_1, \tilde{L}_2, \ldots)$ have the same joint distribution. Also, let $\tilde{M}_k^n$ be the maximum value of $\tilde{L}_j$ over the nodes j at depth k. Then it follows that M_k^n and $\tilde{M}_k^n$ have the same distribution.

Now define further random variables $Z_1, Z_2, \ldots$ from $X_1, X_2, \ldots$ as follows. Let $Z_1 = 1$ and for $i \geq 1$ let $Z_{2i} = X_i Z_i$ and $Z_{2i+1} = (1 - X_i) Z_i$. Then we have $\tilde{L}_i \leq n Z_i$ for each $i = 1, 2, \ldots$. Let Z_k^* be the maximum value of Z_j over the 2^k nodes j at depth k. Then

$$\tilde{M}_k^n \leq n Z_k^* \, .$$

Now the conclusion follows from a series of routine probability inequalities and arguments involving Z_k^*.

The fundamental property of (basic) quicksort that makes the proofs work is given in the following lemma, which may be proved by manipulations of the recurrence (1).

Lemma 2. *Let* n *be a positive integer and let*

$$A_n \ = \ \{n - 1 + q_{k-1} + q_{n-k} - q_n \ : \ k = 1, 2, \ldots, n\} \, .$$

Then $|x| \leq n$ *for all* $x \in A_n$.

3 Filters, Martingales and Concentration

First we give here a very brief introduction to the language of filters and martingales, focussing on the case when the underlying probability space is finite. (For a much fuller introductions see for example [5] or [20].)

The starting point is a probability space $(\Omega, \mathcal{F}, \mathbf{P})$. Thus Ω is the non-empty set of all 'elementary outcomes', the σ-field $\mathcal{F}$ is the set of events, and $\mathbf{P}$ is the probability measure. In many applications, as here, we may take the underlying set Ω to be finite: let us assume this here for simplicity.

Corresponding to any σ-field $\mathcal{G}$ on Ω there is a partition of Ω into non-empty sets, the *blocks* of the partition, such that the σ-field $\mathcal{G}$ is the collection of all sets which are unions of blocks. Suppose that we have a σ-field $\mathcal{G}$ contained in $\mathcal{F}$. A function on Ω which is constant on the blocks of $\mathcal{G}$ is called *$\mathcal{G}$-measurable.* A *random variable* is an $\mathcal{F}$-measurable real-valued function X defined on Ω. Thus in the case when $\mathcal{F}$ consists of all subsets of Ω, each real-valued function on Ω is a random variable.

The *expectation of X conditional on $\mathcal{G}$*, $\mathbb{E}[X \mid \mathcal{G}]$, is the $\mathcal{G}$-measurable function where the constant value on each block of the corresponding partition is the average value of X on the block. The *supremum* $\sup(X \mid \mathcal{G})$ of X *conditional on $\mathcal{G}$* is the $\mathcal{G}$-measurable function where the constant value on each block is the maximum value of X on that block. The *range of X conditional on $\mathcal{G}$*, $\mathrm{ran}(X \mid \mathcal{G})$, is the $\mathcal{G}$-measurable function $\sup(X \mid \mathcal{G}) + \sup(-X \mid \mathcal{G})$.

A nested increasing sequence $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n$ of σ-fields contained in $\mathcal{F}$ is called a *filter.* This corresponds to a sequence of increasingly refined partitions of Ω, starting with the trivial partition into one block. We may think of the filter as corresponding to acquiring information as time goes on: at time k, we know which block of the partition corresponding to $\mathcal{F}_k$ contains our random elementary outcome ω.

Given a filter as above, a sequence $X_0, X_1, X_2, \ldots$ of random variables is called a *martingale* if $\mathbb{E}[X_{k+1} \mid \mathcal{F}_k] = X_k$ for each $k = 0, 1, \ldots$. Now let X be a random variable and let $X_k = \mathbb{E}(X \mid \mathcal{F}_k)$ for $k = 0, 1, \ldots, n$. Then $X_0, X_1, \ldots, X_n$ is a martingale, with $X_0 = \mathbb{E}X$ and $X_n = X$ if X is $\mathcal{F}_n$-measurable. (This is called Doob's martingale process, and for finite filters all corresponding martingales may be obtained in this way.)

Example 1. Let $\Omega = \{0,1\}^n$, let $\mathcal{F}$ be the collection of all subsets of Ω, let $0 < p < 1$, and for each $\omega = (\omega_1, \ldots, \omega_n)$ let $\mathbf{P}(\{\omega\}) = p^j (1-p)^{n-j}$ where $j = \sum_k \omega_k$. This defines our probability space. For each $k = 1, \ldots, n$ define $Z_k(\omega) = \omega_k$ for each $\omega \in \Omega$. Then $Z_1, \ldots, Z_n$ are independent random variables with $\mathbf{P}(Z_k = 1) = 1 - \mathbf{P}(Z_k = 0) = p$ for each k. Also let $Y_k = Z_k - p$, so that $\mathbb{E}[Y_k] = 0$. Let $X_k = Y_1 + \cdots + Y_k$. Let $\mathcal{F}_k$ be the σ-field corresponding to the partition of Ω into the 2^k blocks $\{\omega \in \Omega : \omega_1 = z_1, \ldots, \omega_k = z_k\}$ for each $(z_1, \ldots, z_k) \in \{0,1\}^k$. Then $\mathbb{E}(X_n \mid \mathcal{F}_k) = X_k$ and $\mathrm{ran}(X_k \mid \mathcal{F}_{k-1}) = \mathrm{ran}(Y_k \mid \mathcal{F}_{k-1}) = 1$ (for each $\omega \in \Omega$).

There is a natural filter here, namely

$$\{\Omega, \emptyset\} = \mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n = \mathcal{F},$$

which corresponds to learning the values of the co-ordinates of ω one after another. The σ-field $\mathcal{F}_k$ is the σ-field generated by the random variables $Z_1, \ldots, Z_k$. Also $\mathbb{E}(X_k \mid \mathcal{F}_{k-1}) = X_{k-1}$, and so the random variables X_k form a martingale.

Example 2. Suppose that quicksort acts on a given input of n distinct keys. Consider the infinite binary tree, with nodes numbered $1, 2, 3, \ldots$ level by level and left to right, as before (down to depth n). Let $X = Q_n$. For each $k = 0, 1, 2, \ldots$ let $\mathcal{F}_k$ be the sigma-field generated by all the events for nodes $1, \ldots, k$, and let $X_k = \mathbb{E}[X \mid \mathcal{F}_k]$. Note that for each $k \geq 1$ the list length L_k at node k is $\mathcal{F}_{k-1}$-measurable. By Lemma 2, for each $k \geq 1$

$$\operatorname{ran}(X_k \mid \mathcal{F}_{k-1}) = \operatorname{ran}(X_k - X_{k-1} \mid \mathcal{F}_{k-1}) \leq 2L_k \qquad \text{(for each } \omega \in \Omega\text{)}.$$

But the sum of the list lengths over all nodes at any given depth is at most n; and so the sum of the values $\operatorname{ran}(X_k \mid \mathcal{F}_{k-1})$ over all nodes k at any given depth is at most n.

Let $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n$ be a filter in $\mathcal{F}$. Consider a random variable X, and let $X_0, X_1, \ldots, X_n$ be the martingale obtained by setting $X_k = \mathbb{E}(X \mid \mathcal{F}_k)$. For $1 \leq k \leq n$, we define the *conditional range* ran_k of X_k given $\mathcal{F}_{k-1}$ by setting $\operatorname{ran}_k = \operatorname{ran}(X_k \mid \mathcal{F}_{k-1})$ (which also is $\operatorname{ran}(X_k - X_{k-1} \mid \mathcal{F}_{k-1})$).

Now we come to the concentration result we shall use to prove the upper bound in Theorem 1. To set this result in context, let us first recall the 'bounded differences inequality' for independent random variables (in two-sided form), see for example Theorem 3.1 of [12].

Lemma 3. *Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ be a family of independent random variables with X_k taking values in a set A_k for each k. Suppose that the real-valued function f defined on $\prod_k A_k$ satisfies*

$$|f(\mathbf{x}) - f(\mathbf{x}')| \leq c_k$$

whenever the vectors $\mathbf{x}$ and $\mathbf{x}'$ differ only in the kth co-ordinate. Then for each $x \geq 0$,

$$\mathbf{P}(|f(\mathbf{X}) - \mathbb{E}f(\mathbf{X})| \geq x) \leq 2\, e^{-2x^2/\sum c_k^2}.$$

Next we state an extension of the above result involving martingales, see for example Theorem 3.13 of [12].

Lemma 4. *Let $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n$ be a filter in $\mathcal{F}$, and let the random variable X be $\mathcal{F}_n$-measurable. Let $X_0, X_1, \ldots, X_n$ be the martingale obtained by setting $X_k = \mathbb{E}(X \mid \mathcal{F}_k)$, Suppose that $a_k \leq X_k - X_{k-1} \leq b_k$ for each k, for suitable constants a_k, b_k. Then for each $x \geq 0$,*

$$\mathbf{P}(|X - \mathbb{E}X| \geq x) \leq 2e^{-2x^2/\sum(b_k - a_k)^2}.$$

Note that the conditional range ran_k here is at most the constant $b_k - a_k$. In fact it will suffice to have an $\mathcal{F}_{k-1}$-measurable upper bound on ran_k. Indeed, there is an extension of the last result to the case when we do not necessarily have upper bounds on each conditional range, but we have an upper bound on the sum of squared conditional ranges.

Lemma 5. *Let $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n$ be a filter in $\mathcal{F}$, and let the random variable X be $\mathcal{F}_n$-measurable. Then for each $x \geq 0$ and $r^2 \geq 0$,*

$$\mathbf{P}\left[(|X - \mathbb{E}X| \geq x) \wedge (R^2 \leq r^2) \right] \leq 2\, e^{-\frac{2x^2}{r^2}}$$

where $R^2 = \sum_{j=1}^{n} \mathrm{ran}_j^2$ is the sum of conditional ranges.

This result follows from [4] or from Theorem 3.14 of [12]. (The theorem in [12] is a 'one-sided' result: to obtain Lemma 5 we consider also $-X$.) From the last lemma we easily obtain the concentration result we shall use to prove Theorem 1.

Lemma 6. *Let $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_n$ be a filter in $\mathcal{F}$, let the random variable X be $\mathcal{F}_n$-measurable, and let $X_j = \mathbb{E}[X \mid \mathcal{F}_{j-1}]$ for each $j = 0, \ldots, n$. Let $0 \leq t \leq n$ and $\tau \geq 0$, and assume that $\sum_{j=1}^{t} \mathrm{ran}_j \leq \tau$. Then for each $x \geq 0$ and $v \geq 0$,*

$$\mathbf{P}\left[(|X - \mathbb{E}X| \geq \tau + x) \wedge (\tilde{R}^2 \leq r^2) \right] \leq 2\, e^{-\frac{2x^2}{r^2}}$$

where $\tilde{R}^2 = \sum_{j=t+1}^{n} \mathrm{ran}_j^2$.

Proof. Since

$$|X_t - \mathbb{E}X| = |\sum_{j=1}^{t} (X_j - X_{j-1})| \leq \sum_{j=1}^{t} |X_j - X_{j-1}| \leq \sum_{j=1}^{t} \mathrm{ran}_j \ \leq \tau$$

it follows that

$$\mathbf{P}\left[(|X - \mathbb{E}X| \geq \tau + x) \wedge (\tilde{R}^2 \leq r^2) \right] \leq \mathbf{P}\left((|X - X_t| \geq x) \wedge (\tilde{R}^2 \leq r^2) \right).$$

We shall show that

$$\mathbf{P}\left[(|X - X_t| \geq x) \wedge (\tilde{R}^2 \leq r^2) \mid \mathcal{F}_t \right] \leq 2\, e^{-\frac{2x^2}{r^2}} \tag{3}$$

from which the lemma will follow easily. Consider then any block $\tilde{\Omega}$ of the partition corresponding to $\mathcal{F}_t$. Let $\tilde{X}$ be the restriction of X to $\tilde{\Omega}$; and for each $j = 0, \ldots, n-t$ let $\tilde{\mathcal{F}}_j$ be the restriction of $\mathcal{F}_{t+j}$ to $\tilde{\Omega}$. Now we may apply Lemma 5 to $\tilde{X}$ and the filter $\tilde{\mathcal{F}}_0 \subseteq \tilde{\mathcal{F}}_1 \subseteq \cdots \subseteq \tilde{\mathcal{F}}_{n-t}$ to see that (3) holds, and we are done.

4 Proof of Upper Bound in Theorem 1

We shall apply Lemma 6. We choose parameters carefully, as in [14] (where more detailed computations are given). It suffices to assume that $\epsilon(n) \geq 2 \ln^{(2)} n / \ln n$, since if $\epsilon > 0$ is less than this value then the upper bound is at least 1 (for large n). Let

$$x = x(n) = \lceil \frac{\epsilon\, n \ln n}{\ln^{(2)} n} \rceil \quad \text{and} \quad k = k(n) = \lfloor 2\epsilon \ln n - 2x/n \rfloor.$$

Also let

$$\alpha = \alpha(n) = \epsilon^2 (\ln^{(2)} n)^{-5} \quad \text{and} \quad k_1 = k_1(n) = \lceil (\ln n)(\ln^{(2)} n) \rceil,$$

and let $r^2 = r^2(n) = 4k_1 \alpha n^2$.

Consider the infinite binary tree, with nodes numbered $1, 2, 3, \ldots$ level by level and left to right, as before. Let $X = Q_n$, let $\mathcal{F}_j$ be the sigma-field generated by all the choices up to node j, and let $X_j = \mathbb{E}[X \mid \mathcal{F}_j]$. In fact we need consider the tree only to depth n, since each node at depth at least n must have an empty list.

The sum of the list lengths over all nodes at any given depth is at most n; and so the sum of the conditional ranges over all nodes at any given depth is at most n, see Example 2. Now $t = 2^{k+1} - 1$ is the last node at depth k. Thus the sum of conditional ranges over the nodes $1, \ldots, t$ is at most τ, where $\tau = kn$; that is, $\sum_{j=1}^{t} \text{ran}_j \leq \tau$. It may be checked that $\tau + x \leq \epsilon\, q_n$ for n sufficiently large, and then

$$\begin{aligned} &\mathbb{P}[\,|Q_n - q_n| \geq \epsilon q_n] \\ &\leq \mathbb{P}[\,|X - \mathbb{E}X| \geq \tau + x] \\ &\leq \mathbb{P}[(|X - \mathbb{E}X| \geq \tau + x) \wedge (\tilde{R}^2 \leq r^2)] + \mathbb{P}[\tilde{R}^2 > r^2] \\ &\leq 2e^{-2x^2/r^2} + \mathbb{P}[\tilde{R}^2 > r^2] \end{aligned}$$

by Lemma 6. The first term is negligibly small, since $x^2/r^2 = \Omega((\ln n)\,(\ln^{(2)} n)^2)$. So it remains to deal with the term $\mathbb{P}[\tilde{R}^2 > r^2]$.

If $0 \leq \ell_i \leq \alpha n$ for each i and $\sum_i \ell_i \leq n$ then $\sum_i \ell_i^2 \leq \alpha n^2$. Hence if $M_k^n \leq \alpha n$ then the sum of the squared conditional ranges over all the nodes at any given depth at least k is at most $4\alpha n^2$. Also, if $M_{k_1}^n < 1$ then all lists at depth k_1 are empty, and so the sum of squared ranges over all nodes at depth at least k_1 is 0. Thus

$$\mathbb{P}[\tilde{R}^2 > r^2] \leq \mathbb{P}[M_k^n > \alpha n] + \mathbb{P}[M_{k_1}^n \geq 1].$$

But $k(n) \geq \ln(1/\alpha)$ for n sufficiently large, so we can apply Lemma 1. We find that $\mathbb{P}[M_k^n \geq \alpha n]$ is as on the right side of the equation in Theorem 1, and $\mathbb{P}[M_{k_1}^n \geq 1]$ is negligibly small; and we are done.

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms, 2nd edn. MIT Press, McGraw Hill (2001)
2. Devroye, L.: A note on the height of binary search trees. Journal of the ACM 33, 489–498 (1986)
3. Durrett, R.: Probability: Theory and Examples, 4th edn. Cambridge University Press (2010)
4. Freedman, D.A.: On tail probabilities for martingales. Ann. Probab. 3, 100–118 (1975)
5. Grimmett, G.R., Stirzaker, D.R.: Probability and Random Processes, 3rd edn. Oxford University Press (2001)

6. Hennequin, P.: Combinatorial analysis of quicksort algorithm. Theoretical Informatics and Applications 23, 317–333 (1989)
7. Hoare, C.A.R.: Partition (algorithm 63), quicksort (algorithm 64), and find (algorithm 65). J. ACM 7, 321–322 (1961)
8. Hoare, C.A.R.: Quicksort. Computer J. 5, 10–15 (1962)
9. Knuth, D.: The Art of Computer Programming vol. 3: Sorting and Searching, 3rd edn. Addison-Wesley (1997)
10. Mahmoud, H.M.: Evolution of Random Search Trees. Wiley, New York (1992)
11. McDiarmid, C.: On the method of bounded differences. In: Siemons, J. (ed.) Surveys in Combinatorics. London Math Society (1989)
12. McDiarmid, C.: Concentration. In: Habib, M., McDiarmid, C., Ramirez, J., Reed, B. (eds.) Probabilistic Methods for Algorithmic Discrete Mathematics, pp. 195–248. Springer (1998)
13. McDiarmid, C., Hayward, R.: Strong concentration for quicksort. In: Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 414–421 (1992)
14. McDiarmid, C., Hayward, R.: Large deviations for Quicksort. J. Algorithms 21, 476–507 (1996)
15. Régnier, M.: A limiting distribution for quicksort. Theoretical Informatics and Applications 23, 335–343 (1989)
16. Rösler, U.: A limit theorem for quicksort. Theoretical Informatics and Applications 25, 85–100 (1991)
17. Sedgewick, R.: Quicksort, 1st edn. Garland Publishing Inc. (1980)
18. Shiryaev, A.N.: Probability, 2nd edn. Graduate Texts in Mathematics, vol. 95. Springer (1996)
19. VanEmden, M.H.: Increasing the efficiency of quicksort. Communications of the ACM 13, 563–567 (1970)
20. Williams, D.: Probability with Martingales. Cambridge University Press (1991)

Recent Results on Howard's Algorithm

Peter Bro Miltersen[⋆]

Department of Computer Science
Aarhus University

Abstract. *Howard's algorithm* is a fifty-year old generally applicable algorithm for sequential decision making in face of uncertainty. It is routinely used in practice in numerous application areas that are so important that they usually go by their acronyms, e.g., OR, AI, and CAV. While Howard's algorithm is generally recognized as fast in practice, until recently, its worst case time complexity was poorly understood. However, a surge of results since 2009 has led us to a much more satisfactory understanding of the worst case time complexity of the algorithm in the various settings in which it applies. In this talk, we shall survey these recent results and the open problems that remains.

Howard's algorithm, a.k.a. policy iteration, policy improvement, strategy iteration or strategy improvement, is a generally applicable algorithm for sequentical decision making, subject to an infinite horizon and in face of uncertainty. The uncertainty might come from random behavior of "nature", from the presence of an adversary, or from both. The algorithm was devised by Ron Howard in his PhD thesis from 1960 [13]. In its most basic incarnation, the algorithm is a procedure for solving *finite state and action space discounted Markov decision processes*. It has since its invention been adapted to solve, e.g., undiscounted Markov decsion processes [1,15], two-player discounted payoff games (turn-based or concurrent) [16], two-player parity games [17], two-player recurrent concurrent mean-payoff games [12], simple stochastic games [2], and concurrent reachability games [2], and even certain well-structured infinite-state versions of many of the above models [4].

Each model in any of the above families is given by a set of *states* S, and for each state $i \in S$, a set of actions A_i at the disposal of the decision maker. The action chosen by the decision maker in the current state determines (perhaps stochastically) which state is the next one and also, in some cases, a reward or penalty to the decision maker. Then, the next state becomes the current one, the decision maker again has to make a choice, and so on and so forth, *forever*,

[⋆] The author acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which this work was performed. The authors also acknowledge support from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council.

i.e, the decisions are made with an infinite horizon. The computational problem we consider is to find, for each state $i \in S$, a single preferred action $a_i \in A_i$ for the decision maker or in some cases a probability distribution π_i on such actions. The family $\pi = (a_i)_{i \in S}$ or $\pi = (\pi_i)_{i \in S}$ is called a *policy* or *strategy* for the decision maker. We seek to find a policy that satisfies some qualitative objective or optimizes some quantitative objective, the nature of which depend on the type of model under consideration. As a side product, we also typically compute a *value* v_i for each state $i \in S$ which measures quantitatively how well the decision maker will do if he adopts the policy for play starting with state i being the current one.

In its basic generic form (which is directly applicable without modification to many but not all of the above-mentioned models), Howard's algorithm works as follows. We start with an arbitrary current policy π and compute the values $v_j, j \in S$ associated with this particular policy. Then, for each state $i \in S$, we make a thought experiment. We *pretend* that the objective is to compute a new policy π' that we will adopt for *just one time step*. That is, we pretend that we will make *just one* action choice according to π', after which we will revert to π. With the values $v_j, j \in S$ associated with π at hand, we are able to find the *optimal* π' under this pretense, by solving a relatively easy "local" optimization problem for each state i, finding the optimal deterministic choice a'_i (or the optimal distribution π'_i) to use in state i, and gluing those choices together to make up a policy[1]. Having computed the new policy π', we forget our pretense and the thought experiment and simply consider π' as a policy to use not just once but throughout play, and make it our current policy, i.e, we replace π with π'. Now we iterate this procedure, either until termination (i.e., $\pi' = \pi$, in which case the current policy is in fact optimal) or until the current values are "satifactory".

The worst case number of iterations performed until termination or until the values are "satisfactory" is a natural measure for the worst case time complexity of Howard's algorithm. During the talk, we shall discuss the following families of recent results on this worst case time complexity:

– For discounted Markov decision processes and discounted turn-based two-player games with a constant discount factor, Howard's algorithm is *very fast* in the worst case. More precisely, the number of iterations before termination is $O(mN \log N)$ where N is the number of states of the process and m the number of actions available in each state. This follows from work by Ye [18] who gave a weaker bound for the case of Markov decision processes and work by Hansen, Miltersen and Zwick [10] who improved the bound and generalized it to the two-player case. An interesting problem is to improve the bound to $O(mN)$, matching a lower bound of Hansen and Zwick [11].

[1] An important detail for ensuring termination or convergence of the algorithm is the following: if the choice a_i (or π_i) of the current policy π is already optimal under the pretense, we stick to it, by letting a'_i equal to a_i (or π'_i equal to π_i) rather than picking some different optimal action.

- Howard's algorithm solves simple stochastic games [3] with few coin toss positions *rather fast*: The algorithm terminates after $O(r2^r N)$ iterations, where r is the number of "coin toss states" and N the total number of states [14].
- For undiscounted Markov Decision Processes, discounted Markov Decision Processes with non-constant discount factor and parity games, Howard's algorithm is *rather slow*: The worst case number of iterations is exponential in the number of states. The first result along these lines was a breakthrough result by [6] for the case of parity games that was adapted by Fearnley [5] to the Markov decision process setting.
- For concurrent reachability games, Howard's algorithm is *very slow*: for some games with N states and m actions per state, Howard's algorithm needs $(1/\epsilon)^{m^{\Omega(N)}}$ iterations to have computed a strategy with values with additive distances at most ϵ to the optimal values [9,7]. This is also an upper bound: $(1/\epsilon)^{m^{O(N)}}$ iterations of Howard's algorithm is sufficient to arrive at such a strategy [7,8]. Rather surprisingly, the only proof of the latter fact that we are aware of uses rather non-trivial real algebraic geometry, including separation bounds on algebraic numbers.

References

1. Blackwell, D.: Discrete dynamic programming. Ann. Math. Stat. 33, 719–726 (1962)
2. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Strategy improvement for concurrent reachability games. In: Third International Conference on the Quantitative Evaluation of Systems, QEST 2006, pp. 291–300. IEEE Computer Society (2006)
3. Condon, A.: The complexity of stochastic games. Information and Computation 96, 203–224 (1992)
4. Etessami, K., Yannakakis, M.: Recursive Concurrent Stochastic Games. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part II. LNCS, vol. 4052, pp. 324–335. Springer, Heidelberg (2006)
5. Fearnley, J.: Exponential Lower Bounds for Policy Iteration. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 551–562. Springer, Heidelberg (2010)
6. Friedmann, O.: An exponential lower bound for the parity game strategy improvement algorithm as we know it. In: Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, Los Angeles, CA, USA, August 11-14, pp. 145–156 (2009)
7. Hansen, K.A., Ibsen-Jensen, R., Miltersen, P.B.: The Complexity of Solving Reachability Games Using Value and Strategy Iteration. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 77–90. Springer, Heidelberg (2011)
8. Hansen, K.A., Koucký, M., Lauritzen, N., Miltersen, P.B., Tsigaridas, E.P.: Exact algorithms for solving stochastic games: extended abstract. In: Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, June 6-8, pp. 205–214. ACM (2011)
9. Hansen, K.A., Koucky, M., Miltersen, P.B.: Winning concurrent reachability games requires doubly exponential patience. In: 24th Annual IEEE Symposium on Logic in Computer Science (LICS 2009), pp. 332–341. IEEE (2009)

10. Hansen, T.D., Miltersen, P.B., Zwick, U.: Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. In: Innovations in Computer Science - ICS 2010, January 7-9, pp. 253–263. Tsinghua University Press, Beijing (2011)
11. Hansen, T.D., Zwick, U.: Lower Bounds for Howard's Algorithm for Finding Minimum Mean-Cost Cycles. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part I. LNCS, vol. 6506, pp. 415–426. Springer, Heidelberg (2010)
12. Hoffman, A.J., Karp, R.M.: On nonterminating stochastic games. Management Science, 359–370 (1966)
13. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press, Cambridge (1960)
14. Ibsen-Jensen, R., Miltersen, P.B.: Solving Simple Stochastic Games with Few Coin Toss Positions. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 636–647. Springer, Heidelberg (2012)
15. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York (1994)
16. Rao, S.S., Chandrasekaran, R., Nair, K.P.K.: Algorithms for discounted games. Journal of Optimization Theory and Applications, 627–637 (1973)
17. Vöge, J., Jurdziński, M.: A Discrete Strategy Improvement Algorithm for Solving Parity Games. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000)
18. Ye,Y.: The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate (2010), www.stanford.edu/~yyye/SimplexMDP4.pdf

Advantage of Quantum Strategies in Random Symmetric XOR Games

Andris Ambainis[⋆], Jānis Iraids[⋆⋆], Dmitry Kravchenko[⋆⋆⋆], and Madars Virza[†]

Faculty of Computing, University of Latvia

Abstract. Non-local games are known as a simple but useful model which is widely used for displaying nonlocal properties of quantum mechanics. In this paper we concentrate on a simple subset of non-local games: multiplayer XOR games with 1-bit inputs and 1-bit outputs which are symmetric w.r.t. permutations of players.

We look at random instances of non-local games from this class. We prove a tight bound for the expected performance on the classical strategies for a random non-local game and provide numerical evidence that quantum strategies achieve better results.

1 Introduction

Non-local games [CHTW04] are studied in quantum information, with the goal of understanding the differences between quantum mechanics and classical physics. An example of non-local game is the CHSH (Clauser-Horne-Shimoni-Holt) game [CHSH69, CHTW04]. This is a game played by two parties against a referee. The referee prepares two uniformly random bits x, y and gives one of them to each of two parties. The two parties cannot communicate but can share common randomness or a common quantum state that is prepared before the beginning of the game. The parties reply by sending bits a and b to the referee. They win if $a \oplus b = x \wedge y$.

The maximum winning probability that can be achieved in the CHSH game is 0.75 classically and $\frac{1}{2} + \frac{1}{2\sqrt{2}} = 0.85...$ quantumly. This is interesting because it provides a simple experiment for testing the validity of quantum mechanics. Assume that we implement the referee and the players by devices so that the communication between the players is clearly excluded. If the experiment is repeated m times and players win substantially more than $0.75m$ times, then the results of experiment can be explained using quantum mechanics but not through classical physics.

More generally, we can study non-local games with N players. The referee prepares inputs $x_1, \ldots, x_N$ by picking $(x_1, \ldots, x_N)$ according to some probability

[⋆] Supported by ESF project 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044.

[⋆⋆] Supported by FP7 FET-Open project QCS.

[⋆⋆⋆] Supported by ESF graduate fellowship.

[†] Since September 2011, at the Department of EECS, Massachussets Institute for Technology. Supported by FP7 FET-Open project QCS during this research.

distribution and sends x_i to the i^{th} player. The i^{th} player replies to sending an answer y_i to the referee. Players win if their answers $y_1, \ldots, y_N$ satisfy some winning condition $P(x_1, \ldots, x_N, y_1, \ldots, y_N)$. Similarly as before, the players cannot communicate but can use shared random bits or a common quantum state that has been prepared before receiving $x_1, \ldots, x_N$ from the referee.

Non-local games have been a very popular research topic (often, under the name of Bell inequalities [Be64]). Many non-local games have been studied and large gaps between classical and quantum winning probabilities have been discovered (e.g., [Me90, BV11, BRSW11]).

In this paper, we study non-local games for which the winning condition $P(x_1, \ldots, x_N, y_1, \ldots, y_N)$ is chosen randomly from some class of possible winning conditions. This direction of study was started by [ABB+12] which studied random XOR games with 2 players (players receive inputs $x_1, x_2 \in \{1, \ldots, m\}$ and provide outputs $y_1, y_2 \in \{0, 1\}$, the winning condition depends on x_1, x_2 and $y_1 \oplus y_2$) and showed that, for a random game in this class, its quantum value[1] is between 1.2011... and 1.5638... times its classical value.

We look at a different class of games: N player symmetric XOR games with binary inputs [AKNR09]. For games in this class, both inputs $x_1, \ldots, x_N$ and outputs $y_1, \ldots, y_N$ are binary. The winning condition may depend on the number of $i : x_i = 1$ and the parity of all outputs $\oplus_{i=1}^{N} y_i$. This class of non-local games contains some games with a big quantum advantage. For example, Mermin-Ardehali inequality [Me90, Ar92] can be recast into an N-partite symmetric XOR game whose quantum value is $2^{\lceil \frac{N}{2} - 1 \rceil - \frac{1}{2}}$ times bigger than its classical value.

We show that quantum strategies have some advantage even for a random non-local game from this class:

– We present results of computer experiments that clearly indicate an advantage for quantum strategies.
– We prove that the expected classical value of a random N-player symmetric XOR game is $\frac{0.8475...}{\sqrt[4]{N}}$.
– We provide a non-rigorous argument that the quantum value is $\Omega(\frac{\sqrt{\log N}}{\sqrt[4]{N}})$, with a high probability.

This quantum advantage is, however, much smaller than the maximum advantage achieved by the Mermin-Ardehali game.

2 Definitions

We consider non-local games of N players. We assume that players are informed about the rules of the game and are allowed to have a preliminary discussion. When the game is started, players receive a uniformly random input $x_1, \ldots, x_N \in$

[1] Quantum (classical) value of a game is the maximum difference between the winning probability and the losing probability that can be achieved by quantum (classical) strategies.

$\{0,1\}$ with the i^{th} player receiving x_i. The i^{th} player then produces an output $y_i \in \{0,1\}$. No communication is allowed between the players but they can use shared randomness (in the classical case) or quantum entanglement (in the quantum case).

In an XOR game, the winning condition $P(x_1, \ldots, x_N, y_1, \ldots, y_N)$ depends only on $x_1, \ldots, x_N$ and the overall parity of all the output bits $\oplus_{i=1}^N y_i$. A game is symmetric if the winning condition does not change if $x_1, \ldots, x_N$ are permuted.

If an XOR game is symmetric, the winning condition depends only on $\sum_{i=1}^N x_i$ and $\oplus_{i=1}^N y_i$. Thus, we can define the rules of a game as a sequence of $N+1$ bits $G = G_0 G_1 \ldots G_N$, where G_j defines a "winning" value of $\oplus_{i=1}^N y_i$ for input with $\sum_{i=1}^N x_i = j$.

By a random N-player symmetric XOR game we shall mean a game defined by the $(N+1)$-bit string $G = G_0 G_1 \ldots G_N$ picked uniformly at random.

Let $\Pr_{win}(S)$ and $\Pr_{loss}(S)$ be the probabilities that players win (lose) the game when playing according to a strategy S (which can be either a classical or a quantum strategy). The *value of a strategy* S for a game is $Val\,(S) = \Pr_{win}(S) - \Pr_{loss}(S)$.The *value of a game* $Val\,(G) = \max_S Val\,(S)$ is the value of an optimal (or a best) strategy for this game.

We use Val_C to denote the classical value (maximum of the value over classical strategies) and Val_Q to denote the quantum value (maximum of the value over classical strategies). We omit the subscript C or Q if it is clear from the context whether we are considering quantum or classical value.

3 Optimal Strategies

3.1 Classical Games

Without a loss of generality, we can assume that in a classical game all players use deterministic strategies. (If a randomized strategy is used, we can fix the random bits to the values that achieve the biggest winning probability. Then, a randomized strategy becomes deterministic.)

Then, each player has four different choices — (00), (01), (10), (11). (The first bit here represents the answer on input 0, and second bit represents the answer on input 1. Thus, (ab) denotes a choice to answer a on input 0 and answer b on input 1.)

We use $(00)^{k_0}\,(01)^{k_1}\,(10)^{k_2}\,(11)^{k_3}$ to denote a strategy for N players in which k_0 players use (00), k_1 players use (01), k_2 players use (10) and k_3 players use (11).

Let S be an arbitrary strategy for N players. If exactly one of the players inverts his choice of a strategy bitwise (e.g. (11) $\to$ (00), or (10) $\to$ (01)), this leads to the parity of output bits $\oplus_{i=1}^N y_i$ always being opposite compared to the original strategy. Hence, if $\overline{S}$ is the new strategy, then $\overline{S}$ wins whenever S loses and $\overline{S}$ loses whenever S wins. Therefore,

$$Val\,(S) = \Pr_{win}(S) - \Pr_{loss}(S) = \Pr_{loss}(\overline{S}) - \Pr_{win}(\overline{S}) = -Val\left(\overline{S}\right).$$

From now on, we consider such strategies S and $\overline{S}$ (with exactly one player's choice bitwise inverted) together with a positive value $|Val\,(S)| = \left|Val\,\left(\overline{S}\right)\right|$.

Theorem 1. *[AKNR09] Let S be any classical strategy for a symmetric XOR game with binary inputs. Then, $Val(S)$ is the same as the value of one of $N+1$ following strategies:*

$$(00)^k\,(01)^{N-k}\,,\text{where } k \in \{0, 1, \ldots, N\}. \tag{1}$$

This allows to restrict the set of strategies considerably. In fact, the most useful strategies are $(00)^N$ and $(01)^N$. In our computer experiments, one of these strategies is optimal for $\approx 99\%$ symmetric XOR games. Our rigorous results in Section 5 imply that asymptotically (in the limit of large N) the fraction of games for which one of these strategies is optimal is $1 - o(1)$.

3.2 Quantum Games

The setting in quantum game is identical to the classical one except that the players are allowed to use a quantum system which is potentally entangled before the start of the game. There are two notable results that significantly help the further analysis.

Firstly, it was shown by Werner and Wolf in [WW01] and [WW01a] that in the more general setting where games are not necessarily symmetric the value of the game is described by a simple expression. Let the game be specified by

$$c_{x_1,x_2,\ldots,x_N} = \begin{cases} 1, & \text{if players win when } y_1 \oplus \ldots \oplus y_N = 1 \\ -1, & \text{if players win when } y_1 \oplus \ldots \oplus y_N = 0 \end{cases}$$

Theorem 2

$$Val_Q(G) = \max_{\substack{\lambda_1,\ldots,\lambda_N \in \mathbb{C},\\ |\lambda_1|=\ldots=|\lambda_N|=1}} \left| \sum_{x_1,\ldots,x_N \in \{0,1\}} \frac{c_{x_1,\ldots,x_N} \lambda_1^{x_1} \cdots \lambda_N^{x_N}}{2^N} \right|$$

Secondly, for symmetric games the expression was simplified further in [AKNR09]. Denoting again for convenience $c_j = (-1)^{G_j}$:

Theorem 3

$$Val_Q(G) = \max_{\substack{\lambda \in \mathbb{C},\\ |\lambda|=1}} \left| \sum_{j=0}^{N} \frac{c_j \binom{N}{j} \lambda^j}{2^N} \right|$$

4 Computer Experiments

On the ground of Theorems 1 and 3 we have built efficient optimization algorithms in order to show the difference between classical and quantum versions of symmetric XOR games.

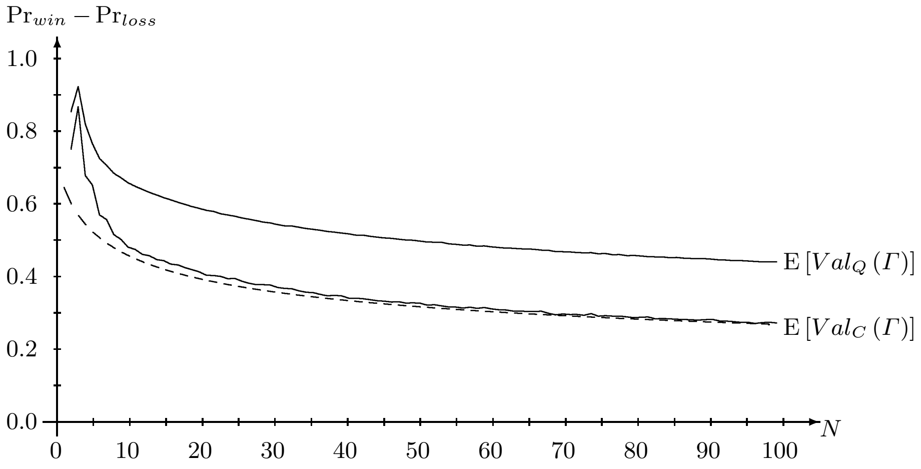


Fig. 1. Expected values of quantum and classical XOR games + classical bound

The Figure 1 shows the expected classical and quantum values for a randomly chosen symmetric XOR game with binary inputs, with the number of players N ranging between 2 and 101[2]. Dashed graph corresponds to the function $f(N) = \frac{0.8475...}{\sqrt[4]{N}}$ derived from Theorem 4.

We see that there is a consistent quantum advantage for all N.

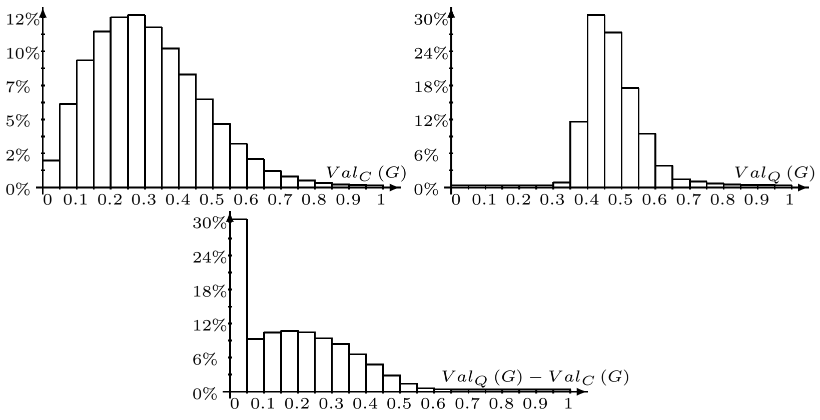


Fig. 2. Histograms of values of random 64-player symmetric XOR games

In the Figure 2, we provide some statistical data on the distribution of the game values (from 10^6 randomly selected games for $N = 64$). The first two

[2] For $N \leq 16$ graphs are precise, as for small number of players it was possible to analyze all 2^{N+1} games. For $N > 16$ we picked 10^5 games at random for each N.

histograms show the distribution of classical and quantum values, respectively. The last one shows the distribution of biases between the values of classical and quantum versions of a game.

We see that the quantum value of a game is more sharply concentrated than the classical value. There is a substantial number (around 30%) of games which have no quantum advantage (or almost no quantum advantage)[3]. For the remaining games, the gap between quantum and classical values is quite uniformly distributed over a large interval.

5 Bounding Classical Game Value

5.1 Results

In this section we first obtain a tight bound on the value of strategies $(00)^N$ and $(01)^N$.

Theorem 4. *For a random N-player symmetric XOR game with binary inputs,*

$$\mathrm{E}\left[\max\left(\left|Val\left((00)^N\right)\right|,\left|Val\left((01)^N\right)\right|\right)\right]=\frac{0.8475...+o(1)}{\sqrt[4]{N}}.$$

We then show that any other strategy from (1) gives a weaker result, with a high probability.

Theorem 5. *For any $c>0$,*

$$\Pr\left[\max_{k:1\le k\le N-1}\left|Val\left((00)^k(01)^{N-k}\right)\right|\ge\frac{c}{\sqrt[4]{N}}\right]=O\left(\frac{1}{N}\right).$$

5.2 Proof of Theorem 4

We first consider the strategy $(00)^N$. As all players always answer 0, the value of this strategy is equal to

$$Val\left((00)^N\right)=\sum_{j=0}^{N}\frac{(-1)^{G_j}\binom{N}{j}}{2^N} \tag{2}$$

For the strategy $(01)^N$, we have

$$Val\left((01)^N\right)=\left|\sum_{j=0}^{N}\frac{(-1)^{G_j+j}\binom{N}{j}}{2^N}\right| \tag{3}$$

We need to find a bound for

$$\mathrm{E}\left[\max\left(\left|Val\left((00)^N\right)\right|,\left|Val\left((01)^N\right)\right|\right)\right]$$

[3] However, our results in the next sections indicate that the fraction of such games will tend to 0 for larger N.

$$= \mathrm{E}\left[\max\left(\left|\sum_{j=0}^{N}\frac{(-1)^{G_j}\binom{N}{j}}{2^N}\right|,\left|\sum_{j=0}^{N}\frac{(-1)^{G_j+j}\binom{N}{j}}{2^N}\right|\right)\right] \tag{4}$$

Among the summands of these two sums let us first evaluate those which are equal for both of sums, i.e. for even j's, and then for remaining summands, which have opposite values in these sums, i.e. for odd j's:

$$\mathrm{E}\left[\max\left(\left|Val\left((00)^N\right)\right|,\left|Val\left((01)^N\right)\right|\right)\right]$$

$$= \mathrm{E}\left[\max\left(\pm\sum_{\substack{0\le j\le N,\\ j\text{ is even}}}\frac{(-1)^{G_j}\binom{N}{j}}{2^N}\pm\sum_{\substack{0\le j\le N,\\ j\text{ is odd}}}\frac{(-1)^{G_j}\binom{N}{j}}{2^N}\right)\right]$$

$$= \mathrm{E}\left[\left|\sum_{\substack{0\le j\le N,\\ j\text{ is even}}}\frac{(-1)^{G_j}\binom{N}{j}}{2^N}\right|\right]+\mathrm{E}\left[\left|\sum_{\substack{0\le j\le N,\\ j\text{ is odd}}}\frac{(-1)^{G_j}\binom{N}{j}}{2^N}\right|\right] \tag{5}$$

Let Σ_{even} and Σ_{odd} be the two sums in (5). Then, we have

$$\mathrm{Var}\left[\Sigma_{even}\right] = \sum_{\substack{0\le j\le N,\\ j\text{ is even}}}\left(\frac{\binom{N}{j}}{2^N}\right)^2 = \frac{\binom{2N}{N}}{2\cdot 4^N}.$$

Similarly, $\mathrm{Var}\left[\Sigma_{odd}\right] = \frac{\binom{2N}{N}}{2\cdot 4^N}$. From the central limit theorem, in the limit of large N, each of random variables Σ_{even} and Σ_{odd} can be approximated by a normally distributed random variable with the same mean (which is 0) and variance. If X is a normally distributed random variable with $E[X]=0$, then $E[|X|]=\sqrt{\frac{2}{\pi}}\sqrt{\mathrm{Var}[X]}$. Hence, (5) is equal to

$$\mathrm{E}\left[|\Sigma_{even}|\right]+\mathrm{E}\left[|\Sigma_{odd}|\right]$$

$$= \sqrt{\frac{2}{\pi}}(1+o(1))\sqrt{\mathrm{Var}\left[\Sigma_{even}\right]}+\sqrt{\frac{2}{\pi}}(1+o(1))\sqrt{\mathrm{Var}\left[\Sigma_{odd}\right]}$$

$$= (1+o(1))\sqrt{\frac{2}{\pi}}\sqrt{\frac{2\binom{2N}{N}}{4^N}} = (1+o(1))\sqrt[4]{\frac{16}{\pi^3}}\frac{1}{\sqrt[4]{N}} = \frac{0.8475...+o(1)}{\sqrt[4]{N}}$$

where the second-to-last equality follows from the approximation of the binomial coefficients $\binom{2N}{N}=(1+o(1))\frac{4^N}{\sqrt{\pi N}}$.

5.3 Variance of Other Strategies

We now consider other strategies S, with the goal of proving Theorem 5. To do that, we first compute the variances for their values $Val(S)$. (Our goal is to prove Theorem 5 by Chebyshev's inequality, which we do in the next subsection.)

We start with the variance of $Val((00)^N)$. Because of (2), the variance of $Val((00)^N)$ can be calculated as

$$\mathrm{Var}\left[Val\left((00)^N\right)\right] = \sum_{j=0}^{N} \mathrm{Var}\left[\frac{(-1)^{G_j}\binom{N}{j}}{2^N}\right] = \sum_{j=0}^{N} \frac{\binom{N}{j}^2}{4^N} = \frac{\binom{2N}{N}}{4^N} \approx \frac{1}{\sqrt{\pi N}} \qquad (6)$$

We note that $Val\left((00)^N\right)$ for the game $G_0G_1G_2G_3G_4\dots$ is exactly the same as $Val\left((01)^N\right)$ for the game $G_0\overline{G_1}G_2\overline{G_3}G_4\dots$, with all odd bits inverted.

More generally, assume that we have a strategy $(00)^k(01)^{N-k}$. We can invert the answers by all players for the case $x_i = 1$. Then, the overall parity of answers $\oplus_{i=1}^N y_i$ stays the same if an even number of players have received $x_i = 1$ and changes to opposite value if an odd number of players have received $x_i = 1$. If we simultaneously invert all odd-numbered bits G_i in the winning condition, the game value does not change. From this, we conclude that for each k,

$$Val\left((00)^k(01)^{N-k}\right) = Val\left((00)^{N-k}(01)^k\right) \qquad (7)$$

We now consider the value for the second strategy from (1), $(00)^{N-1}(01)$, for a random symmetric XOR game.

Probability distribution of $\oplus_{i=0}^N y_i$ when $\sum_{i=0}^N x_i = j$ is the following:

j	0	1	2	3	4	5	6	7	...
$\Pr\left[\oplus_{i=0}^N y_i = 0\right]$	1	$\frac{1}{N}$	$\frac{N-2}{N}$	$\frac{3}{N}$	$\frac{N-4}{N}$	$\frac{5}{N}$	$\frac{N-6}{N}$	$\frac{7}{N}$	...
$\Pr\left[\oplus_{i=0}^N y_i = 1\right]$	0	$\frac{N-1}{N}$	$\frac{2}{N}$	$\frac{N-3}{N}$	$\frac{4}{N}$	$\frac{N-5}{N}$	$\frac{6}{N}$	$\frac{N-7}{N}$	...

Given input with $\sum_{i=0}^N x_i = j$, the strategy outputs even or odd answer, depending on whether or not the last player has received 1, i.e. with probabilities $\frac{N-j}{N}$ and $\frac{j}{N}$ (unlike 1 and 0 in the case of symmetric strategies).

Therefore, variance of the strategy $(00)^{N-1}(01)$ for input with $\sum_{i=0}^N x_i = j$ is $\left(\frac{N-j}{N} - \frac{j}{N}\right)^2 = \left(\frac{N-2j}{N}\right)^2$.

Summing up variances for all possible j's, we get

$$\begin{aligned}\mathrm{Var}\left[Val\left((00)^{N-1}(01)\right)\right] &= \sum_{j=0}^{N} \mathrm{Var}\left[\frac{\pm\frac{N-2j}{N}\binom{N}{j}}{2^N}\right] \\ &= \sum_{j=0}^{N}\left(\frac{\pm\frac{N-2j}{N}\binom{N}{j}}{2^N}\right)^2 = \frac{\binom{2N}{N}}{4^N(2N-1)} \approx \frac{1}{\sqrt{\pi N}(2N-1)},\end{aligned} \qquad (8)$$

with the third equality following from Lemma 1 in the appendix (Notation $\pm\frac{N-2j}{N}$ denotes a random variable with equiprobable values $+\frac{N-2j}{N}$ and $-\frac{N-2j}{N}$.)

Due to (7), the value of strategy $(00)(01)^{N-1}$ has the same variance.

Other strategies of type $(00)^{N-k}(01)^k$ where $2 \leq k \leq N-2$ have much smaller variances, which can be expressed as follows:

$$\begin{aligned}\operatorname{Var}\left[Val\left((00)^{N-k}(01)^k\right)\right] &= \sum_{j=0}^{N}\left(\frac{\left(\sum_{l=0}^{j}(-1)^l\frac{\binom{k}{l}\binom{N-k}{j-l}}{\binom{N}{j}}\right)\binom{N}{j}}{2^N}\right)^2 \\ &= \frac{\sum_{j=0}^{N}\left(\sum_{l=0}^{j}(-1)^l\binom{k}{l}\binom{N-k}{j-l}\right)^2}{4^N}\end{aligned} \tag{9}$$

The expression $\sum_{l=0}^{j}(-1)^l\binom{k}{l}\binom{N-k}{j-l}$ inside (9) is well known Kravchuk polynomial $K_j(k)$, whose square can be bounded by strict inequality provided in [Kr01]:

$$\left(\sum_{l=0}^{j}(-1)^l\binom{k}{l}\binom{N-k}{j-l}\right)^2 = (K_j(k))^2 < 2^N\binom{N}{j}\binom{N}{k}^{-1}. \tag{10}$$

This inequality implies that

$$\begin{aligned}\operatorname{Var}\left[Val\left((00)^{N-k}(01)^k\right)\right] &= \frac{\sum_{j=0}^{N}(K_j(k))^2}{4^N} \\ &< \frac{\sum_{j=0}^{N}2^N\binom{N}{j}\binom{N}{k}^{-1}}{4^N} = \binom{N}{k}^{-1}.\end{aligned} \tag{11}$$

5.4 Proof of Theorem 5

Among the strategies $(00)^k(01)^{N-k}$, $k \in \{1,\ldots,N-1\}$, two strategies (for $k=1$ and $k=N-1$) have variance $\approx \frac{1}{\sqrt{\pi N}(2N-1)}$, and the remaining $N-3$ strategies have variance less than $\frac{1}{\binom{N}{k}}$.

We now apply Chebyshev inequality, using those two bounds on the variance. We have

$$\begin{aligned}&\Pr\left[\left|Val\left((00)^{N-1}(01)\right)\right| \geq \frac{\lambda}{\sqrt[4]{\pi N}\sqrt{2N-1}}\right] \\ &\qquad = \Pr\left[\left|Val\left((00)(01)^{N-1}\right)\right| \geq \frac{\lambda}{\sqrt[4]{\pi N}\sqrt{2N-1}}\right] \leq \frac{1}{\lambda^2},\end{aligned}$$

$$\text{and, for } 2 \leq k \leq N-2:$$
$$\Pr\left[\left|Val\left((00)^{N-k}(01)^k\right)\right| \geq \frac{\lambda}{\sqrt{\binom{N}{k}}}\right] \leq \frac{1}{\lambda^2}. \tag{12}$$

We now combine the bounds (12) into one upper bound.

$$\Pr\left[\max_{0\le k\le N}\left|Val\left((00)^{N-k}(01)^{k}\right)\right|\ge B\right]\le$$

$$\le 2\times\frac{1}{\left(B\sqrt[4]{\pi N}\sqrt{(2N-1)}\right)^2}+\sum_{k=2}^{N-2}\frac{1}{\left(B\sqrt{\binom{N}{k}}\right)^2}$$

$$=\frac{2}{B^2\sqrt{\pi N}\,(2N-1)}+O\left(\frac{1}{B^2N^3}\right),\quad (13)$$

with the last equality following from $\binom{N}{k}\ge\binom{N}{2}$ and the fact that we are summing over $N-3$ values for k: $k\in\{2,\ldots,N-2\}$. Taking $B=\frac{c}{\sqrt[4]{N}}$ proves theorem 5.

6 Bounding Quantum Game Value

So far we have not been able to find a tight lower bound on the mean value of the game in quantum case. However, we provide some insights which could lead to a solution to the problem. We can bound the value from below by

$$\max_{\substack{\lambda\in\mathbb{C},\\|\lambda|=1}}\left|\sum_{j=0}^{N}\frac{c_j\binom{N}{j}\lambda^j}{2^N}\right|\ge\max_{\alpha}\left|\sum_{j=0}^{N}\frac{c_j\binom{N}{j}\cos(\alpha j)}{2^N}\right|\quad (14)$$

The sum $\sum_{j=0}^{N}c_j\cos(\alpha j)$ where c_j are indepedent random variables with mean 0 and variance 1 has been extensively studied under the name "random trigonometric polynomials" by Salem and Zygmund[SZ54] and others. Their results imply that there exist constants A and B such that

$$\lim_{M\to\infty}\Pr\left[A\sqrt{M\log M}\le\max_{\alpha}\left|\sum_{j=0}^{M}c_j\cos(\alpha j)\right|\le B\sqrt{M\log M}\right]=1\quad (15)$$

To apply (15), the crucial step is to reduce a sum $c_j\binom{N}{j}\cos(\alpha j)$ with binomial coefficients to a sum $c_j\cos(\alpha j)$ not containing binomial coefficients.

We propose a following non-rigorous approximation. We first drop the terms with $j\le\frac{N}{2}-\sqrt{N}$ and $j\ge\frac{N}{2}-\sqrt{N}$. For the remaining terms, we replace $\binom{N}{j}$ with $\binom{N}{N/2}$ (since $\binom{N}{j}=\Theta(\binom{N}{N/2})$ for $j\in[\frac{N}{2}-\sqrt{N},\frac{N}{2}+\sqrt{N}]$). If this approximation can be justified, it reduces (14) to (15) with $M=2\sqrt{N}$. This would lead to a lower bound of

$$E_G[Val_Q(G)]=\Omega\left(\frac{\sqrt{\log M}}{\sqrt{M}}\right)=\Omega\left(\frac{\sqrt{\log N}}{\sqrt[4]{N}}\right).$$

We are currently working on making this argument rigorous.

7 Conclusion

We studied random instances of symmetric N-player XOR games with binary inputs, obtaining tight bounds for the classical value of such games. We also presented a non-rigorous argument bounding the quantum value. Our results indicate that quantum strategies are better than classical for random games in this class, by a factor of $\Omega(\sqrt{\log N})$. An immediate open problem is to make our bound for quantum strategies precise, by bounding the error introduced by our approximations.

A more general question is: can we analyze random instances of other classes of non-local games? We currently know how to analyze random games for 2-player XOR games with N-valued inputs and for symmetric N-player games with binary inputs.

Can we analyze, for example, 3-player XOR games with N-valued inputs? In a recent work, Briët and Vidick [BV11] have shown big gaps between quantum and classical strategies for this class of games. However, methods of analyzing such games are much less developed and this makes analysis of random games quite challenging. Developing tools for it is an interesting direction for future work.

References

[ABB+12] Ambainis, A., Bačkurs, A., Balodis, K., Kravčenko, D., Ozols, R., Smotrovs, J., Virza, M.: Quantum Strategies Are Better Than Classical in Almost Any XOR Game. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012. LNCS, vol. 7391, pp. 25–37. Springer, Heidelberg (2012)

[AKNR09] Ambainis, A., Kravchenko, D., Nahimovs, N., Rivosh, A.: Nonlocal Quantum XOR Games for Large Number of Players. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 72–83. Springer, Heidelberg (2010)

[Ar92] Ardehali, M.: Bell inequalities with a magnitude of violation that grows exponentially with the number of particles. Physical Review A 46, 5375–5378 (1992)

[Be64] Bell, J.: On the Einstein-Podolsky-Rosen paradox. Physics 1(3), 195–200 (1964)

[BV11] Briët, J., Vidick, T.: Explicit lower and upper bounds on the entangled value of multiplayer XOR games. arXiv:1108.5647

[BRSW11] Buhrman, H., Regev, O., Scarpa, G., de Wolf, R.: Near-Optimal and Explicit Bell Inequality Violations. In: Proceedings of CCC 2011, pp. 157–166 (2011)

[CHSH69] Clauser, J., Horne, M., Shimony, A., Holt, R.: Physical Review Letters 23, 880 (1969)

[CHTW04] Cleve, R., Høyer, P., Toner, B., Watrous, J.: Consequences and limits of nonlocal strategies. In: Proceedings of CCC 2004, pp. 236–249 (2004); Also quant-ph/0404076

[Kr01] Krasikov, I.: Nonnegative Quadratic Forms and Bounds on Orthogonal Polynomials. Journal of Approximation Theory 111, 31–49 (2001), doi:10.1006/jath.2001.3570

[Me90] Mermin, D.: Extreme Quantum Entanglement in a Superposition of Macroscopically Distinct States. Physical Review Letters 65(15) (1990)

[SZ54] Salem, R., Zygmund, A.: Some properties of trigonometric series whose terms have random signs. Acta Math. 91, 245–301 (1954)

[WW01] Werner, R.F., Wolf, M.M.: Bell inequalities and Entanglement. Quantum Information and Computation 1(3), 1–25 (2001)

[WW01a] Werner, R.F., Wolf, M.M.: All multipartite Bell correlation inequalities for two dichotomic observables per site. Physical Review A 64, 32112 (2001)

A Appendix

Lemma 1

$$\sum_{j=0}^{N}\left(\frac{N-2j}{N}\binom{N}{j}\right)^2 = \frac{\binom{2N}{N}}{2N-1}$$

Proof

$$\begin{aligned}
&\sum_{j=0}^{N}\left(\frac{N-2j}{N}\binom{N}{j}\right)^2\\
&=\sum_{j=0}^{N}\left(1-4\frac{j}{N}+4\frac{j^2}{N^2}\right)\binom{N}{j}^2\\
&=\sum_{j=0}^{N}\binom{N}{j}^2-4\sum_{j=1}^{N}\binom{N-1}{j-1}\binom{N}{j}+4\sum_{j=1}^{N}\binom{N-1}{j-1}^2\\
&=\binom{2N}{N}-4\binom{2N-1}{N-1}+4\binom{2N-2}{N-1}\\
&=\binom{2N}{N}-2\binom{2N}{N}+\frac{2N}{2N-1}\binom{2N}{N}\\
&=\frac{\binom{2N}{N}}{2N-1}
\end{aligned}$$

Verification of Liveness Properties on Closed Timed-Arc Petri Nets[⋆]

Mathias Andersen, Heine Gatten Larsen, Jiří Srba, Mathias Grund Sørensen, and Jakob Haahr Taankvist

Department of Computer Science, Aalborg University, Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark

Abstract. Verification of closed timed models by explicit state-space exploration methods is an alternative to the wide-spread symbolic techniques based on difference bound matrices (DBMs). A few experiments found in the literature confirm that for the reachability analysis of timed automata explicit techniques can compete with DBM-based algorithms, at least for situations where the constants used in the models are relatively small. To the best of our knowledge, the explicit methods have not yet been employed in the verification of liveness properties in Petri net models extended with time. We present an algorithm for liveness analysis of closed Timed-Arc Petri Nets (TAPN) extended with weights, transport arcs, inhibitor arcs and age invariants and prove its correctness. The algorithm computes optimized maximum constants for each place in the net that bound the size of the reachable state-space. We document the efficiency of the algorithm by experiments comparing its performance with the state-of-the-art model checker UPPAAL.

1 Introduction

TAPAAL [7] is a an efficient, open-source tool for modelling and verification of Timed-Arc Petri Nets (TAPN) extended with transport/inhibitor arcs and age invariants. The timing information (age) is attached to tokens and intervals on input arcs restrict the ages of tokens suitable for transition firing. The verification techniques implemented in the tool include four different translations to UPPAAL timed automata [11], supporting both reachability and liveness properties, and its own verification engine for reachability analysis. The actual verification in any of those approaches rely on searching the abstracted state-space represented via zones and using the data structure called Difference Bound Matrix (DBM) [8].

Unfortunately, for the verification of liveness questions, neither of the methods return error traces (loops in this case) with concrete time delays and not all requested features, like weighted arcs, are currently supported. As in the general case with both open and closed intervals the concrete error traces do not necessarily form a finite loop, we restrict ourselves to the large and practically relevant

[⋆] This work is partially supported by the research center IDEA4CPS.

subclass of TAPNs with only closed intervals. It is a folklore result that the continuous and discrete-time semantics coincide on the class of closed systems (see e.g. [6]). In nowadays tools the discretization is not sufficiently exploited, perhaps due to its simplicity as remarked by Lamport [12]. Nevertheless, a few existing studies show that discretization of the state-space may be beneficial [6,14,12], at least in the situations with sufficiently small constants that appear in the model.

We suggest an efficient algorithm for verification of liveness properties on closed TAPNs extended with weighted transport/inhibitor arcs and age invariants. The main contributions include a detailed analysis of the maximum constants individually computed for each place in the net, the complete proof of soundness and completeness of the proposed algorithm and last but not least an efficient C++ implementation and its full integration into the model checker TAPAAL. The efficiency is documented by experiments ranging from standard academic examples for testing the performance of tools like Fischer's protocol for mutual exclusion to larger case-studies from the health-care domain. We compare the CPU-time performance of our discretized algorithm with the efficient DBM-based engines, including the state-of-the-art model checker UPPAAL [1]. The main conclusion is that our algorithm can outperform the DBM-based methods as long as the constants in the model are not too large. Moreover, the discrete method scales very well in the size of the problems, allowing us to solve problems with constants that grow more than linearly while increasing the problem size. As a bonus, our algorithm always returns loop-like counter examples with concrete time delays, a feature that allows the user to easily debug possible design flaws in the models.

Related Work. Lamport [12], Bozga et al. [6], Beyer [4,3] and Popova-Zeugmann [14] present different methods for discrete model checking of timed systems. The first three papers present explicit methods for the model of timed automata, while the third one studies discretization for Time Petri Nets (TPN), a Petri net model that is substantially different from ours (timing information is attached to transitions and not to tokens like in TAPNs). While reachability is in general undecidable for TAPNs [16], a time-bounded reachability for TAPNs with discrete semantics was shown decidable in [15]. The technique, however, does not apply for verification of liveness properties because we search here for the presence of an infinite execution satisfying certain invariant properties and such executions are often time-diverging. Our liveness algorithm is instead parameterized by a number k allowing us to explore markings with at most k tokens (after the removal of dead tokens that cannot be used for transition firing). In case of bounded nets it always provides conclusive answers while for unbounded nets (where the liveness problem is undecidable) the answer is conclusive only in the cases where a loop (counter example) can be found among markings with at most k tokens (the number k is a part of the input).

To the best of our knowledge, there are no published experiments for discrete verification of liveness properties on TAPNs, moreover extended with the additional modelling features that require a nontrivial static analysis in order to

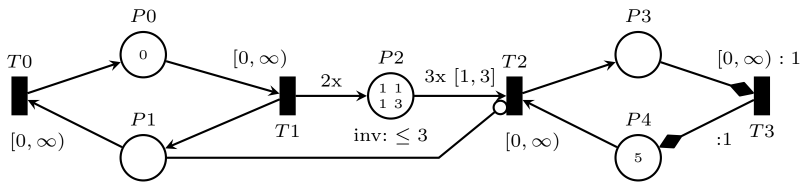


Fig. 1. Producer-Consumer Example

minimize the size of maximum constants relative to the individual places in the net. We assess the performance of our approach by performing a comparison against the state-of-the-art model checker UPPAAL and the results are encouraging as documented in Section 5.

2 Timed-Arc Petri Nets

Let us first informally introduce the TAPN model. Figure 1 shows an example of a producer-consumer system. The circles represent places and rectangles represent transitions. A marking of the net is given by the distribution of timed tokens; in our case there is one token of age 0 in $P0$, three tokens of age 1 and one of age 3 in $P2$ and one token of age 5 in $P4$.

Places and transitions are connected by arcs and input arcs to transitions are labelled by time intervals. The arc from $T1$ to $P2$ has the weight 2, denoted by 2x, meaning that two tokens will be produced by firing the transition. Similarly the arc from $P2$ to $T2$ has the weight 3, meaning that three tokens of age between 1 and 3 must be consumed when firing $T2$, while at the same time there may not be any token in place $P1$ (denoted by the inhibitor arc with the circle head). In our example the transition $T2$ can fire, consuming three tokens from the place $P2$ (these can be either $\{1,1,1\}$ or $\{1,1,3\}$) and one token from place $P4$, while depositing a new token of age 0 to the place $P3$. The pair of arcs from $P3$ to $P4$ with a diamond head are called transport arcs and they always come in pairs (in our example with the index :1). They behave like normal arcs but when a token is consumed in $P3$ and produced to $P4$, its age is preserved. Places can also have age invariants like the one denoted by "inv: ≤ 3" in the place $P2$. This restricts the maximum age of tokens present in such places. In our example, there is already a token of age 3 in $P2$, meaning that we cannot wait any more and are without any delay forced to fire some transition.

Let us now give a formal definition of the TAPN model. Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ and $\mathbb{N}_0^\infty = \mathbb{N}_0 \cup \{\infty\}$. A *discrete timed transition system* (DTTS) is a triple $(S, Act, \rightarrow)$ where S is the set of states, Act is the set of actions and $\rightarrow \subseteq S \times (Act \cup \mathbb{N}_0) \times S$ is the transition relation written as $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$. If $a \in Act$ then we call it a *switch transition*, if $a \in \mathbb{N}_0$ we call it a *delay transition*. By $\rightarrow^*$ we denote the reflexive and transitive closure of the relation $\rightarrow \overset{\text{def}}{=} \bigcup_{a \in Act} \xrightarrow{a} \cup \bigcup_{d \in \mathbb{N}_0} \xrightarrow{d}$.

We define the set of *well-formed time intervals* as $\mathcal{I} \overset{\text{def}}{=} \{[a,b] \mid a \in \mathbb{N}_0, b \in \mathbb{N}_0^\infty, a \leq b\}$ and a subset of $\mathcal{I}$ used in invariants as $\mathcal{I}^{\text{inv}} = \{[0,b] \mid b \in \mathbb{N}_0^\infty\}$. For an interval $[a,b]$ we define $[a,b]_L = a$ and $[a,b]_R = b$ in order to denote the lower and upper bound of the interval, respectively. Let $maxBound(I)$ denote the largest bound different from infinity in the interval I, formally $maxBound([a,b]) = a$ if $b = \infty$, and $maxBound([a,b]) = b$ otherwise.

We can now define the closed TAPN model with weighted arcs.

Definition 1 (Closed Timed-Arc Petri Net). *A closed TAPN is an 8-tuple $N = (P, T, IA, OA, g, w, Type, I)$ where*

- *P is a finite set of* places*,*
- *T is a finite set of* transitions *such that $P \cap T = \emptyset$,*
- *$IA \subseteq P \times T$ is a finite set of* input arcs*,*
- *$OA \subseteq T \times P$ is a finite set of* output arcs*,*
- *$g : IA \to \mathcal{I}$ is a* time constraint function *assigning guards to input arcs,*
- *$w : IA \cup OA \to \mathbb{N}$ is a function assigning* weights *to input and output arcs,*
- *$Type : IA \cup OA \to \boldsymbol{Types}$ is a* type function *assigning a type to all arcs, where $\boldsymbol{Types} = \{Normal, Inhib\} \cup \{Transport_j \mid i \in \mathbb{N}\}$ such that*
  - *if $Type(a) = Inhib$ then $a \in IA$,*
  - *if $Type((p,t)) = Transport_j$ for some $(p,t) \in IA$ then there is exactly one $(t,p') \in OA$ such that $Type((t,p')) = Transport_j$ and $w((p,t)) = w((t,p'))$,*
  - *if $Type((t,p')) = Transport_j$ for some $(t,p') \in OA$ then there is exactly one $(p,t) \in IA$ such that $Type((p,t)) = Transport_j$ and $w((p,t)) = w((t,p'))$,*
- *$I : P \to \mathcal{I}^{inv}$ is a function assigning* age invariants *to places.*

The preset of input places of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p,t) \in IA, Type((p,t)) \neq Inhib\}$. Similarly, the postset of output places of t is defined as $t^\bullet = \{p \in P \mid (t,p) \in OA\}$.

Let $N = (P, T, IA, OA, g, w, Type, I)$ be a TAPN and let $\mathcal{B}(\mathbb{N}_0)$ be the set of all finite multisets over $\mathbb{N}_0$. A *marking* M on N is a function $M : P \longrightarrow \mathcal{B}(\mathbb{N}_0)$ where for every place $p \in P$ and every token $x \in M(p)$ we have $x \in I(p)$. The set of all markings over N is denoted by $\mathcal{M}(N)$.

We use the notation (p,x) to denote a token at a place p with the age $x \in \mathbb{N}_0$. We write $M = \{(p_1,x_1),(p_2,x_2),\ldots,(p_n,x_n)\}$ for a marking with n tokens of ages x_i located at places p_i and we define $size(M) = \sum_{p \in P} |M(p)|$. A marked TAPN (N, M_0) is a TAPN N together with an initial marking M_0 with all tokens of age 0.

Definition 2 (Enabledness). *Let $N = (P, T, IA, OA, g, w, Type, I)$ be a TAPN. We say that a transition $t \in T$ is* enabled *in a marking M by the multisets of tokens $In = \{(p, x_p^1), (p, x_p^2), \ldots, (p, x_p^{w((p,t))}) \mid p \in {}^\bullet t\} \subseteq M$ and $Out = \{(p', x_{p'}^1), (p', x_{p'}^2), \ldots, (p', x_{p'}^{w((t,p'))}) \mid p' \in t^\bullet\}$ if*

- *for all input arcs except the inhibitor arcs the tokens from In satisfy the age guards of the arcs, i.e.*

$$\forall (p,t) \in \mathit{IA}.\,\mathit{Type}((p,t)) \neq \mathit{Inhib} \Rightarrow x_p^i \in g((p,t)) \text{ for } 1 \leq i \leq w((p,t))$$

- *for any inhibitor arc pointing from a place p to the transition t, the number of tokens in p satisfying the guard is smaller than the weight of the arc, i.e.*

$$\forall (p,t) \in \mathit{IA}.\,\mathit{Type}((p,t)) = \mathit{Inhib} \Rightarrow |\{x \in M(p) \mid x \in g((p,t))\}| < w((p,t))$$

- *for all input arcs and output arcs which constitute a transport arc the age of the input token must be equal to the age of the output token and satisfy the invariant of the output place, i.e.*

$$\forall (p,t) \in \mathit{IA}. \forall (t,p') \in \mathit{OA}.\,\mathit{Type}((p,t)) = \mathit{Type}((t,p')) = \mathit{Transport}_j$$
$$\Rightarrow \left(x_p^i = x_{p'}^i \wedge x_{p'}^i \in I(p')\right) \text{ for } 1 \leq i \leq w((p,t)).$$

- *for all output arcs that are not part of a transport arc the age of the output token is 0, i.e.*

$$\forall (t,p') \in \mathit{OA}.\,\mathit{Type}((t,p') = \mathit{Normal} \Rightarrow x_{p'}^i = 0 \text{ for } 1 \leq i \leq w((p,t)).$$

In Figure 1 the transition $T2$ is enabled by $\mathit{In} = \{(P2,1),(P2,1),(P2,1),(P4,5)\}$ and $\mathit{Out} = \{(P3,0)\}$. As the tokens in the place $P2$ have different ages, $T2$ is also enabled by an alternative multiset of tokens $\mathit{In} = \{(P2,1),(P2,1),(P2,3),(P4,5)\}$.

A given TAPN $N = (P,T,\mathit{IA},\mathit{OA},g,w,\mathit{Type},I)$ defines a DTTS $T(N) \overset{\text{def}}{=} (\mathcal{M}(N),T,\rightarrow)$ where states are the markings and the transitions are as follows.

- If $t \in T$ is enabled in a marking M by the multisets of tokens In and Out then t can be *fire* and produce the marking $M' = (M \smallsetminus \mathit{In}) \uplus \mathit{Out}$ where $\uplus$ is the multiset sum operator and $\smallsetminus$ is the multiset difference operator; we write $M \xrightarrow{t} M'$ for this switch transition.
- A *time delay* $d \in \mathbb{N}$ is allowed in M if $(x+d) \in \mathit{Inv}(p)$ for all $p \in P$ and all $x \in M(p)$, i.e. by delaying d time units no token violates any of the age invariants. By delaying d time units in M we reach the marking M' defined as $M'(p) = \{x + d \mid x \in M(p)\}$ for all $p \in P$; we write $M \xrightarrow{d} M'$ for this delay transition.

A computation of the net $M_0 \rightarrow M_1 \rightarrow \cdots \rightarrow M_n$ is denoted by $\{M_i\}_{i=0}^n$ and we call it a *run*. If the sequence is infinite, we write $\{M_i\}_{i\geq 0}$.

2.1 Liveness Verification Problem

The liveness question asks about the existence of a maximal run where every marking satisfies some proposition referring to the distribution of tokens. For that purpose let the set of propositions Φ be given by the abstract syntax $\varphi ::=$

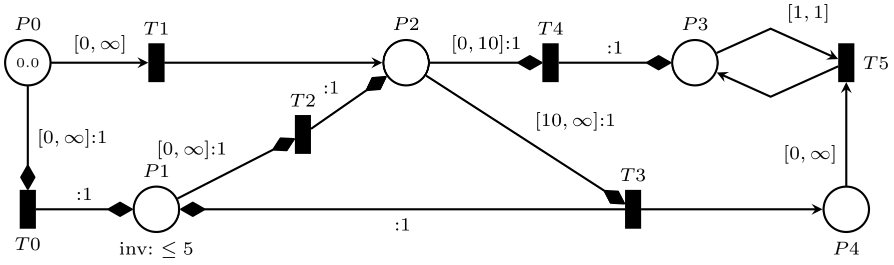


Fig. 2. Example of TAPN

$p \bowtie n \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi$ where $\bowtie \in \{\leq, <, =, \neq, \geq >\}$, $p \in P$ and $n \in \mathbb{N}_0$. The satisfaction relation $M \models \varphi$ is defined in the expected way where $M \models p \bowtie n$ iff $|M(p)| \bowtie n$.

Given a TAPN (N, M_0), a *maximal run* is either an infinite run $\{M_i\}_{i\geq 0}$ or a finite run $\{M_i\}_{i=0}^{n}$ with $M_n \nrightarrow$, meaning that M_n does not allow for any switch or positive-delay transition. A maximal run (finite or infinite) is denoted by $\{M_i\}$.

Definition 3 (The Liveness Problem ($EG\varphi$)). *Given a marked TAPN (N, M_0) and a proposition $\varphi \in \Phi$, the liveness problem is to decide whether there is a maximal run $\{M_i\}$ starting in M_0 such that $M_i \models \varphi$ for all i.*

We can define the standard dual operator AF by $AF\varphi \overset{\text{def}}{=} \neg EG\neg\varphi$, meaning that eventually the property φ will be satisfied on any execution of the net.

3 State-Space Reduction

The state-space of TAPNs is infinite in two dimensions: the number of tokens can be unbounded and the ages of tokens range over natural numbers. Indeed, the model (extended with inhibitor arcs and age invariants) has the full Turing power (see e.g. [16,10]). In order to enable automatic verification, we restrict ourselves to bounded TAPNs where the maximum number of tokens in any reachable marking is a priori bounded by some constant k. For restricting the ages of tokens, we do not need to remember the concrete ages of tokens in a place that are older than the maximum constant relevant for that place. This idea was suggested in [16,9] for the basic TAPN model without any additional features. We shall now refine the technique for the more general class of extended TAPNs that contain age invariants, transport and inhibitor arcs and we further enhance it with the reduction of dead tokens in order to optimize its performance.

To motivate the technical definitions that follow, let us consider the net in Figure 2. Note that in place $P3$ the relevant ages of tokens are 0 and 1. Any token of age 2 or more cannot be used for transition firing and can be safely removed from the net. We shall call $P3$ the *dead-token* place with the maximum constant 1. Any place that contains an invariant, like $P1$ in our example, will fall into the category *invariant* places and the maximum constant will be the

$$C_{arc}((p,t)) = \begin{cases} min([I(p')]_R, [g((p,t))]_R) & \text{if } \mathit{Type}((p,t)) = \mathit{Transport}_j, \\ & \mathit{Type}((t,p')) = \mathit{Transport}_j, \text{ and} \\ & I(p') \neq [0,\infty] \\ maxBound(g((p,t))) & \text{otherwise} \end{cases} \tag{1}$$

$$C_{place}(p) = \begin{cases} [I(p)]_R & \text{if } [I(p)]_R \neq \infty \\ -1 & \text{if } I(p) = [0,\infty] \text{ and} \\ & \forall (p,t) \in \mathit{IA}.(g((p,t)) = [0,\infty] \\ \max_{(p,t)\in \mathit{IA}}(C_{arc}((p,t))) & \text{otherwise.} \end{cases} \tag{2}$$

Fig. 3. Definitions of C_{arc} and C_{place}

upper-bound of the respective invariant. Clearly, no tokens can be older that this constant but at the same time we may not just remove the tokens that cannot be used for any transition firing as they could restrict delay transitions in the future. The remaining places are called *standard* places meaning that instead of the ages of tokens that exceed the maximum constant for a given place, we only need to remember how many there are, but not their exact ages. For example in the place $P0$ all outgoing arcs have the guard $[0,\infty]$, so it may look like that we only need to remember the number of tokens, not their ages. Indeed, if there were no transport arcs this would be the case. However, in our example the pair of transport arcs moving tokens to $P1$ increase the maximum constant for the place $P0$ to 5 as the concrete age is relevant up to this number in order to avoid breaking of the age invariant in $P1$. In general, there might be a series of transport arcs that can influence the maximum constant for a place and we show how to optimize such constants to be as small as possible while still preserving the liveness property we want to verify.

We start by the definition of *causality* function. The causality function finds the causality set of places that are linked with the place p by a chain of transport arcs with the right endpoints of the guard intervals equal to ∞.

Let $p \in P$. The set $cau(p)$ is the smallest set of places such that

- $p \in cau(p)$, and
- if $p' \in cau(p)$ and $(p',t) \in \mathit{IA}, (t,p'') \in \mathit{OA}$ such that $\mathit{Type}(p',t) = \mathit{Transport}_j$, $\mathit{Type}(t,p'') = \mathit{Transport}_j$ with $[g((p',t))]_R = \infty$ and $I(p') = \infty$ then $p'' \in cau(p)$.

In the net from Figure 2 we get that for example $cau(P0) = \{P0, P1\}$, $cau(P1) = \{P1\}$ and $cau(P2) = \{P2, P1\}$.

Next we define the maximum relevant constants for input arcs by Equation (1) in Figure 3 as a function $C_{arc} : \mathit{IA} \to \mathbb{N}_0$. The first case deals with the situation when the arc is a transport arc that moves tokens to a place with a nontrivial age invariant; here it is enough to consider the minimum of the invariant upper-bound and the largest constant in the guard different from infinity. If this is not the case, we consider just the maximum bound in the guard.

Arc	Type	C_{arc}
$P0 \to T0$	$\mathit{Transport}_1$	5
$P0 \to T1$	*Normal*	0
$P1 \to T2$	$\mathit{Transport}_1$	0
$P2 \to T3$	$\mathit{Transport}_1$	10
$P2 \to T4$	$\mathit{Transport}_1$	10

Place	C_{place}	C_{max}	*cat*
$P0$	-1	5	*Std*
$P1$	5	5	*Inv*
$P2$	10	10	*Std*
$P3$	1	1	*Dead*
$P4$	-1	-1	*Std*

Fig. 4. Calculation of C_{arc}, C_{place}, C_{max} and *cat* for the TAPN in Figure 2

The constant of a place (without considering any causality) is defined by Equation (2) in Figure 3 as a function $C_{place} : P \to \mathbb{N}_0 \cup \{-1\}$. The constant is either the upper-bound of a nontrivial age invariant in the place, or -1 if all arcs from p only have trivial guards; in this case we do not care about the ages of the tokens in p. Otherwise the constant for p is the largest constant of any arc starting at p.

We are now ready to divide places into three categories and compute the maximum relevant constants taking into account the causality set of places. In liveness verification, the query will also influence the category of places, so we consider the function $\mathit{Places} : \Phi \to \mathcal{P}(P)$ that for a given proposition φ returns the set of places that syntactically appear in φ. We can now calculate the function $C_{max} : P \to \mathbb{N}_0 \cup \{-1\}$ returning the maximum constant for each place (taking into account also the transport arcs) and the function $\mathit{cat} : P \to \{\mathit{Inv}, \mathit{Dead}, \mathit{Std}\}$ returning the category for each place $p \in P$ as follows.

– If $I(p) \neq [0, \infty]$ then $C_{max}(p) = [I(p)]_R$ and $\mathit{cat}(p) = \mathit{Inv}$.
– Otherwise $C_{max}(p) = \max\{C_{place}(p') \mid p' \in \mathit{cau}(p)\}$ and if either
  (i) there is $t \in T$ such that $(p, t) \in \mathit{IA}$ and $\mathit{Type}((p, t)) = \mathit{Inhib}$), or
  (ii) there is $t \in T$ such that $(p, t) \in \mathit{IA}$ and $[g((p, t))]_R = \infty]$, or
  (iii) $p \in \mathit{Places}(\varphi)$)
  then $\mathit{cat}(p) = \mathit{Std}$, else $\mathit{cat}(p) = \mathit{Dead}$.

The conditions (i)–(iii) list all situations where we are not allowed to remove tokens above the maximum constant as the concrete number of these tokens is relevant for the behaviour of the net or for the the proposition φ. An example of the calculation of C_{max} and *cat* is given in Figure 4, assuming $\mathit{Places}(\varphi) = \{P1\}$.

3.1 Bounded Marking Equivalence

Given the maximum constants and categories of places, we can now define an equivalence relation on markings that will have a finite number of equivalence classes and can be used in the liveness checking algorithm.

Let C_{max} and *cat* be computed as above and let M be a marking. We split M into two markings $M_>$ and $M_\leq$ as follows: $M_>(p) = \{x \in M(p) \mid x > C_{max}(p)\}$ and $M_\leq(p) = \{x \in M(p) \mid x \leq C_{max}(p)\}$ for all places $p \in P$. Clearly, $M = M_> \uplus M_\leq$.

Definition 4 (Bounded Marking Equivalence). *Let M and M' be markings on a TAPN N. We say that M and M' are equivalent, written $M \equiv M'$, if*

– $M_{\leq}(p) = M'_{\leq}(p)$ *for all* $p \in P$, *and*
– $|M_{>}(p)| = |M'_{>}(p)|$ *for all* $p \in P$ *where* $cat(p) = Std$.

The equivalence relation implies that in *Dead* places we do not care about the tokens with ages greater than C_{max} and that in *Std* places we do not care about tokens with ages greater than C_{max}, as long as there are equally many of them in both markings. An important correctness argument is the fact that that the relation $\equiv$ is a timed bisimulation where delays on one side and matched by exactly the same delays on the other side (see e.g. [13]). The proof is done by a detailed case analysis and can be found in the full version of the paper.

Theorem 1. *The relation $\equiv$ is a timed bisimulation.*

In order to calculate a representative marking for each $\equiv$-equivalence class, we define the function *cut* and present Lemma 1 that is proved in the full version of the paper.

Definition 5 (Cut). *The function $cut : \mathcal{M}(N) \rightarrow \mathcal{M}(N)$ is given by*

$$cut(M)(p) = \begin{cases} M_{\leq}(p) & \text{if } cat(p) \in \{Inv, Dead\} \\ M_{\leq}(p) \uplus \{ \underbrace{C_{max}(p)+1, \ldots, C_{max}(p)+1}_{|M_{>}(p)| \text{ times}} \} & \text{if } cat(p)=Std \end{cases}$$

for all $p \in P$. We call the marking $cut(M)$ canonical.

Lemma 1 (Properties of Canonical Markings)

1. *For any marking M we have $M \equiv cut(M)$.*
2. *Given two markings M_1 and M_2 if $M_1 \equiv M_2$ then $cut(M_1) = cut(M_2)$.*
3. *Let M be a marking and $\varphi \in \Phi$ be a proposition then $M \models \varphi$ iff $cut(M) \models \varphi$.*

4 Liveness Algorithm

We can now present Algorithm 1 answering the liveness verification problem. It is essentially a depth-first search algorithm where the *Waiting* stack stores the currently unexplored successors that satisfy the invariant property φ. In the *Trace* stack we keep track of the run from the initial marking to the currently explored marking. A counter recording the number of unexplored successors for each marking on the *Trace* stack is used for the coordination between the *Trace* and *Waiting* stacks. The main loop contains a boolean variable indicating whether the current marking is the end of a maximal run (in case no further successors exist). If this is the case, the algorithm terminates as a maximal run satisfying φ has been found. Otherwise new canonical successors (by transition firing and one-unit delay) are added by calling the function *AddToPW*, making

```
1  input: A TAPN $(N, M_0)$, proposition $\varphi \in \Phi$ and $k \in \mathbb{N}$ s.t. $size(cut(M_0)) \leq k$.
2  output: True if there is a maximal run $\{M_i\}$ s.t. $M_i \models \varphi$ and
   $size(cut(M_i)) \leq k$, false otherwise.
3  begin
4      $Passed := \emptyset$; $Waiting.InitStack()$; $Trace.InitStack()$; $M'_0 := cut(M_0)$;
5      if $M'_0 \models \varphi$ then
6          $Waiting.push(M'_0)$;
7      while $\neg Waiting.isEmpty()$ do
8          $M := Waiting.pop()$;
9          if $M \notin Passed$ then
10             $Passed := Passed \cup \{M\}$; $M.successors$:=0;
11             $Trace.push(M)$; $endOfMaxRun$ := true;
12             foreach $M'$ s.t. $M \xrightarrow{t} M'$ do
13                 AddToPW($M$,$M'$); $endOfMaxRun$ := false;
14             if $\min_{(p,x)\in M}([I(p)]_R - x) > 0$ then
15                 AddToPW($M$,$M'$) where $M \xrightarrow{1} M'$; $endOfMaxRun$ := false;
16             if $endOfMaxRun$ then
17                 return true      /* terminate and return the Trace stack */ ;
18         else
19             $Trace.top().successors--$
20         while $\neg Trace.isEmpty() \wedge Trace.top().successors = 0$ do
21             $Trace.pop()$;
22             if $Trace.isEmpty()$ then
23                 return false      /* terminate the whole algorithm *.;
24             $Trace.top().successors--$;
25     return false;
26 AddToPW($M$,$M'$): begin
27     $M'' := cut(M')$;
28     if $M'' \in Trace$ then
29         return true      /* terminate and return the loop on the Trace stack */;
30     if $M'' \notin Passed \wedge M'' \models \varphi \wedge size(M'') \leq k$ then
31         $Waiting.push(M'')$;
32         $M.successors{+}{+}$;
```

Algorithm 1. Liveness algorithm

sure that only markings that satisfy φ are added to the *Waiting* list. The function also checks for the presence of a loop on the *Trace* stack, in which case the algorithm terminates and returns true. A bound k is also an input to the algorithm, making sure that only canonical markings with no more than k tokens are explored during the search. If the net is k-bounded, this has no effect on the actual search. For unbounded nets, our algorithm still terminates and provides a suitable under-approximation of the net behaviour, giving conclusive answers if a loop is found and inconclusive answers otherwise.

Table 1. Fischer's protocol scaled by the number of processes (rows) and the size of maximum constant (columns). First line is a native UPPAAL model, second line is the fastest translation to timed automata and using the UPPAAL engine, and third line is our discrete TAPAAL engine. The symbol ◷ stands for more than 900 seconds.

Processes \ Constants	3	5	7	9	11	13	15
5	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.1	0.1	0.3	0.7	1.8	3.7	7.9
6	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	0.1	**0.1**	0.5	1.8	5.3	13.3	29.6
7	4.6	4.6	4.6	4.6	4.6	4.6	4.6
	47.5	47.2	47.0	47.1	47.2	47.4	47.1
	0.1	**0.2**	**1.1**	**4.5**	14.4	40.7	99.3
8	422.5	422.6	421.5	422.4	421.9	422.1	422.3
	◷	◷	◷	◷	◷	◷	◷
	0.1	**0.4**	**2.4**	**10.5**	**37.8**	**115.2**	**309.8**
9	◷	◷	◷	◷	◷	◷	◷
	◷	◷	◷	◷	◷	◷	◷
	0.1	**0.7**	**4.5**	**22.4**	**90.5**	**300.4**	**888.2**
10	◷	◷	◷	◷	◷	◷	◷
	◷	◷	◷	◷	◷	◷	◷
	0.1	**1.1**	**8.2**	**45.9**	**202.2**	**733.5**	◷

Table 2. Blood transfusion case study scaled by the number of patients; time is seconds

Patients	Translations	TAPAAL
1	0.11	0.04
2	28.08	0.93
3	>5400.00	30.47
4	>5400.00	1072.50

Theorem 2 (Correctness). *Let TAPN (N, M_0) be a closed TAPN, $\varphi \in \Phi$ a proposition and $k \in \mathbb{N}$ a number such that $size(cut(M_0)) \leq k$. Algorithm 1 terminates, and it returns true if there is a maximal run $\{M_i\}$ such that $M_i \models \varphi$ and $size(cut(M_i)) \leq k$ and false otherwise.*

Proof (sketch). Termination follows from the fact that we only store markings after applying the function *cut*, giving us together with at most k tokens in the net a finite state-space. The soundness and completeness part of Theorem 2 rely on Lemma 1 and Theorem 1 and details are given the full version of the paper. $\square$

5 Experiments

The liveness algorithm has been implemented and fully integrated into the verification tool TAPAAL [7] and it can be downloaded (as a beta-release) from http://www.tapaal.net. We performed a number of experiments[1] and due to the space limitation mention only two of them. The results of verification of Fischer's algorithm for mutual exclusion are given in Table 1. We asked here an EG query checking whether there is an infinite schedule allowing us to repeatedly enter the critical section within a given time interval. The query is not satisfied and hence the whole state-space where the proposition holds is searched. The table shows the verification times for a native UPPAAL model of the protocol (first line), the best time for a translation (see [7] for details) to timed automata and then using the UPPAAL engine (second line) and our discretized algorithm (third line). The gray cells mark the experiments where our method was the fastest one. The reader can observe that the DBM-based methods in the first two lines are immune to scaling of constants. On the other hand, our algorithm scales significantly better with increasing the number of processes. Hence for larger instances, we can handle larger and larger constants while still outperforming the DBM-based methods. In fact, the size of the constants we can deal with for the given time limit grows more than linearly as we increase the number of processes. We have observed similar behaviour in other case studies too, like e.g. in the Lynch-Shavit protocol that is presented in the full version of the paper.

In order to test the performance on a realistic case-study, we verified soundness (AF query) of a blood transfusion medical workflow (details can be found in [2]) where the maximum constant is of size 90 and it considerably outperforms the translation approach verified via UPPAAL engine. Results are given in Table 2 and we compare our engine with the fastest translation to UPPAAL timed automata.

6 Conclusion

We presented a discrete algorithm for verification of liveness properties on extended timed-arc Petri nets and provided its implementation and integration into the model checker TAPAAL. The main technical contribution is the partitioning of the places in the net to three categories and an optimized computation of the individual maximum constants, allowing us to design an efficient loop detection algorithm based on depth-first search. We proved the algorithm correct and demonstrated on several examples its applicability as an alternative to DBM-based search algorithms. The techniques can be easily adapted to work also for reachability analysis.

Our approach is well suited for larger models with relatively small constants. Due to an on-the-fly removal of dead tokens that appear in the net, we were able to successfully verify models that are in general unbounded and where DBM-based methods give inconclusive answers (for example in case of the Alternating Bit Protocol (ABP) with perfect communication channels presented as the standard example in the TAPAAL distribution). In the future work we shall focus

[1] We report here the data obtained on MacBook Pro 2.7GHz INTEL Core i7 with 8 GB RAM and 64-bit versions of UPPAAL and TAPAAL.

on space-optimization of the proposed technique, on a symbolic computation of the delay operator and on comparing the method to BDD-based state space exploration (as exploited e.g. in the tool Rabbit [5]).

References

1. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: QEST 2006, pp. 125–126 (September 2006)
2. Bertolini, C., Liu, Z., Srba, J.: Verification of timed healthcare workflows using component timed-arc Petri nets. In: FHIES 2012. Springer (to appear, 2012)
3. Beyer, D.: Efficient Reachability Analysis and Refinement Checking of Timed Automata Using BDDs. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 86–91. Springer, Heidelberg (2001)
4. Beyer, D.: Improvements in BDD-Based Reachability Analysis of Timed Automata. In: Oliveira, J.N., Zave, P. (eds.) FME 2001. LNCS, vol. 2021, pp. 318–343. Springer, Heidelberg (2001)
5. Beyer, D., Lewerentz, C., Noack, A.: Rabbit: A Tool for BDD-Based Verification of Real-Time Systems. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 122–125. Springer, Heidelberg (2003)
6. Bozga, M., Maler, O., Tripakis, S.: Efficient Verification of Timed Automata Using Dense and Discrete Time Semantics. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 125–141. Springer, Heidelberg (1999)
7. David, A., Jacobsen, L., Jacobsen, M., Jørgensen, K.Y., Møller, M.H., Srba, J.: TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 492–497. Springer, Heidelberg (2012)
8. Dill, D.L.: Timing Assumptions and Verification of Finite-state Concurrent Systems. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1990)
9. Hanisch, H.M.: Analysis of Place/transition Nets with Timed Arcs and its Application to Batch Process Control. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 282–299. Springer, Heidelberg (1993)
10. Jacobsen, L., Jacobsen, M., Møller, M.H.: Undecidability of Coverability and Boundedness for Timed-Arc Petri Nets with Invariants. In: Proc. of MEMICS 2009, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2009)
11. Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: Verification of timed-arc Petri nets. In: SOFSEM 2011, pp. 46–72 (2011)
12. Lamport, L.: Real-Time Model Checking Is Really Simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
13. Larsen, K.G., Wang, Y.: Time-abstracted bisimulation: Implicit specifications and decidability. Information and Computation 134(2), 75–101 (1997)
14. Popova-zeugmann, L.: Essential states in time Petri nets. Informatik-Berichte 96 (1998)
15. de Frutos Escrig, D., Ruiz, V.V., Marroquín Alonso, O.: Decidability of Properties of Timed-Arc Petri Nets. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 187–206. Springer, Heidelberg (2000)
16. Ruiz, V.V., Cuartero Gomez, F., de Frutos Escrig, D.: On non-decidability of reachability for timed-arc Petri nets. In: Proceedings of the 8th International Workshop on Petri Net and Performance Models (PNPM 1999), pp. 188–196 (1999)

Fast Algorithm for Rank-Width

Martin Beyß

RWTH Aachen University

Abstract. Inspired by the heuristic algorithm for boolean-width by Telle et. al. [1], we develop a heuristic algorithm for rank-width. We compare results on graphs of practical relevance to the established bounds of boolean-width. While the width of most graphs is lower than the known values for tree-width, it turns out that the boolean-width heuristic is able to find decompositions of significantly lower width. In a second step we therefore present a further algorithm that can decide if for a graph G and a value k exists a rank-decomposition of width lower than k. This enables to show that boolean-width is in fact lower than or equal to rank-width on many of the investigated graphs.

1 Introduction

Many interesting problems on graphs like TSP or Hamiltonian path are *NP*-complete. Thus, there is no fast way to solve them in general unless of course $P = NP$. Width parameters of graphs like tree-width, clique-width or rank-width can be used to construct fixed parameter tractable (*FPT*) algorithms for many of these problems. This means if the input graphs are restricted to have width at most k, a solution can be calculated in time at most $f(k) \cdot p(n)$ where f is a computable function, n the size of the graph and p a polynomial independent of k. For small k this provides a realistic chance of solving these problems even on large graphs. Many of those algorithms use dynamic programming and need a decomposition of the graph. Hence, practical methods of calculating decompositions of low width are of crucial importance.

A major result is Courcelles theorem [2] which states that every graph problem that is expressible as a MSO_2 formula can be decided in linear time for graphs of fixed tree-width. A similar theorem exists for rank-width and clique-width and a formula in MSO_1 [3]. Recent research shows that these theorems are not just of theoretical interest but also of practical applicability [4]. Accordingly, an algorithm that is able to calculate rank-decompositions gives a possibility to solve many hard problems on graphs. As [5] shows, there are several algorithms that can compute reasonably small tree-decompositions, but tree-width is only low for sparse graphs. Consequently, for dense graphs other width-measures have to be considered. An overview of different width-measures other than tree-width and their algorithmic applications can be found, e.g. in [6].

Although rank-width has been extensively researched over the last years, there are only few practical results. For example in [7], Oum proposed an $\mathcal{O}(|V|^3)$ algorithm that for a fixed k either returns a rank-decomposition of width at most $3k-1$ or confirms that the rank-width is larger than k. Nevertheless, no implementation is known and actual rank-decompositions only exist for very few graphs.

In [1] Hvidevold et. al. presented a novel approach to find decompositions of low boolean-width even on dense graphs. We can reuse their work and develop a heuristic algorithm to calculate rank-decompositions. Moreover, our algorithm is able to build rank-decompositions on graphs for which no bounds of other width-measures are known in particular many graphs where the boolean-width heuristic failed. This reveals the main weakness of boolean-width for practical application. Calculating the boolean-width of a given decomposition is costly especially in comparison with rank-width.

As an extension, an additional algorithm is implemented that decides for a given k if a graph has rank-width less than k. This allows the calculation of lower bounds for rank-width. These lower bounds can provide a quality estimation for the heuristic algorithm. Furthermore, we are able to compare results for boolean-width and rank-width on graphs from real life application, i.e. treewidthLIB [8].

2 Preliminaries

Graphs in this work are simple, undirected and loop free. A graph G is a tuple (V, E) where V is a set and $E \subset [V]^2$ and $[V]^2$ is the set of all two-element subsets of V. An element $v \in V$ is called a *vertex*, $e \in E$ is called *edge*. Instead of writing $v \in V$, we may write $v \in G$; the same goes for $e \in E$. As a shorthand we write $|G|$ for $|V|$ and $\|G\|$ for $|E|$. For $G = (V, E)$ we define functions $V(G) = V$ and $E(G) = E$, which assign a graph to its vertex and edge sets.

For a graph $G = (V, E)$ and a set $V' \subset V$ we define the *induced subgraph* as $G[V'] = (V', E')$ where $E' \subset E$ so that $\{v_1, v_2\} \in E'$ if $v_1, v_2 \in V'$. For a Tree $T = (V_T, E_T)$, $n \in V_T$ is called a *node*.

Tree-width[9] is most likely the best studied and understood width measure for graphs. It measures how similar a graph is to a tree. Hence, a tree has tree-width 1 while a complete graph with n vertices has tree-width $n-1$.

Definition 1. *Let $G = (V, E)$ be a graph, $T = (V_T, E_T)$ a tree and $\mathcal{V} : V_T \to 2^V,\ t \mapsto V_t$ a function. We call $(T, \mathcal{V})$ a* tree-decomposition *of G if it fulfils the following three conditions*

1. *$V = \bigcup_{t \in V_T} \mathcal{V}(t)$*
2. *For each $e = \{u, v\},\ e \in E$ there is a node $t \in V_T$ so that $u, v \in \mathcal{V}(t)$*
3. *For nodes $t_1, t_2, t_3 \in V_T$ where t_2 lies on a path from t_1 to t_3 there is always $\mathcal{V}(t_1) \cap \mathcal{V}(t_3) \subseteq \mathcal{V}(t_2)$*

The width *of a tree-decomposition $(T, \mathcal{V})$ is $max\{|\mathcal{V}(t)|-1\ ;\ t \in V_T\}$. The lowest width of all possible tree-decompositions is the* tree-width *of G or $tw(G)$.*

Branch-width is a width measure that was introduced in [10]. Since its invention it has been generalized, and here the definition of [11] is used and extended to suit our needs.

Definition 2. *Let V be a finite set and $f : 2^V \to \mathbb{R}$ a function. If for any set $X \subseteq V$ f has the property that $f(X) = f(V \backslash X)$, it is* symmetric*.*

A subcubic *tree is a tree in which every node has degree at most three. If T is a subcubic tree and $L : V \to \{v \; ; \; v \text{ is a leaf in } T\}$ a surjective function, then (T, L) is called a* partial branch-decomposition *of f. If f is bijective, (T, L) is a* total branch-decomposition *of f.*

For an edge $e \in T$, $T \backslash \{e\}$ induces a partition (X, Y) of the leaves of T. The value of $f(L^{-1}(X))$ is the width *of the edge e of the partial branch-decomposition (T, L). The maximum width over all edges of T is the* width *of the partial branch-decomposition (T, L). The minimum width over all possible total branch-decompositions of f is called the* branch-width *of f, denoted by $bw(f)$.*

If $f(X) > f(Y)$, we say X is wider *than Y or analogously, Y is* narrower *than Y. Consequently, that term can be applied to branch-decompositions.*

Rank-width[11] is a recent width measure which is built on branch-width.

Definition 3. *Let $G = (V, E)$ be a simple, connected, undirected graph without loops and M its adjacency matrix over $GF(2)$. By M_Y^X we denote the submatrix of M with rows in X and columns in Y, for $X, Y \subseteq V$. For $X \subset V$ we define the* cut-rank *as $cutrk(X) = rk(M_{\overline{X}}^X)$ where rk is the standard matrix rank and $\overline{X}$ is the set complement of X in V.*

Obviously, the cut-rank function is symmetric, for that reason we can define the branch-width of the cut-rank function of a graph G as the rank-width *of G, denoted as $rw(G)$. Analogously, the term* (partial) rank-decomposition *can be defined.*

Like rank-width, boolean-width[12] is the branch-width of a special function, in this case the *cut-bool* function.

Definition 4. *Let $G = (V, E)$ be a graph and let $N(a)$ be the set of vertices adjacent to $a \in G$. For $X \subset V$ we define $UN(X)$ to be the* union of neighbourhoods *of subsets of X in $\overline{X}$.*

$$UN(X) = \{S \subseteq \overline{X} \; ; \; \exists \; A \subseteq X \wedge S = \overline{X} \cap \bigcup_{a \in A} N(a)\} \tag{1}$$

With this the cut-bool *function can be defined.*

$$cutbool : 2^V \to \mathbb{R}, \; X \mapsto \log_2 |UN(X)| \tag{2}$$

The branch-width of the cut-bool function of a graph G is called boolean-width *of G, denoted by $boolw(G)$.*

Remark 1. The set of $GF(2)$-sums of neighbourhoods of subsets of X in $\overline{X}$ is called $SN(X)$.

$$SN(X) = \{S \subseteq \overline{X} \; ; \; \exists \; A \subseteq X \wedge S = \overline{X} \cap \bigwedge_{a \in A} N(a)\} \tag{3}$$

We can easily see ([12])that $cutrk(X) = \log_2 |SN(X)|$, which shows that boolean-width and rank-width are quite closely related.

3 An Upper Bound Algorithm

This Section describes a heuristic algorithm for rank-width, which is an improved version of the boolean-width algorithm presented in [1]. Adapting their algorithm for rank-width is easily possible as both width measures are closely related branch-widths (c.f. Remark 1). However, this algorithm does not carry some of the limitations of its boolean-width variant because the cut-rank function is much easier and faster to calculate than the cut-bool function.

3.1 Overview

The goal of this algorithm is to heuristically find a narrow total rank-decomposition for a given graph. At the beginning an initial rank-decomposition is calculated, and then attempts to improve it are made. In a first step we assume that there already exists a total rank-decomposition $R = (T, L)$.

Algorithm 1. Main loop

```
Input: a graph $G = (V, E)$
Output: a total decomposition $R = (T, L)$ of $G$
 1: Let $T$ be an empty tree and $L : \emptyset \to \emptyset$
 2: $best \leftarrow \infty$
 3: $R \leftarrow (T, L)$
 4: while algorithm should keep running do
 5:     $e = (n_1, n_2) \leftarrow$ first edge of $R$
 6:     ImproveSubtree($n_1$)
 7:     ImproveSubtree($n_2$)
 8:     if $R$ is total then
 9:         $R_{opt} \leftarrow R$                    ▷ new best decomposition
10:         $best \leftarrow$ Width($R$)
11:     else
12:         $R \leftarrow R_{opt}$                    ▷ reset decomp.
13:     end if
14: end while
15: return $R_{opt}$
```

Consequently, we skip the initialisation and start at line 4 of Algorithm 1. Inside the while loop it is tried to improve the given rank-decomposition. ImproveSubtree returns a total decomposition if and only if an improvement could be made. Accordingly, the next steps depend on whether R is total or not. If a better decomposition is found, it is saved along with its width (line 9). Otherwise R is reset. This is repeated for a certain amount of time.

For initialisation an empty decomposition is created. This has to be considered for the ImproveSubtree routine. The first rank-decomposition is calculated greedily while later calculations use a mix of greedy and random decisions.

Algorithm 2. ImproveSubtree

```
Input: a subtree rooted at $n$
Output:
 1: if $n$ is a leaf then
 2:     $(X, Y) \leftarrow$ Split($n$)
 3: else
 4:     $(X, Y) \leftarrow$ RandomSwap($n$)
 5: end if
 6: if $max(cutrk(X), cutrk(Y)) < best$ then
 7:     remove subtrees rooted at $n$ (if any)
 8:     add children $n_1, n_2$ to $n$
 9:     $\forall x \in X : L(x) \leftarrow n_1$, $\forall y \in Y : L(y) \leftarrow n_2$
10: end if
11: if $n$ has children $n_1$ and $n_2$ of width $< best$ then
12:     ImproveSubtree($n_1$), ImproveSubtree($n_2$)
13: end if
```

Algorithm 2 shows the functionality of the ImproveSubtree routine. If the root of the subtree is an internal node, a good split is already known and we try to randomly improve it. Otherwise a good split has to be calculated greedily. Assume that the node n has some children, and thus the RandomSwap routine is called. In Algorithm 2.6 it is now checked if the new cuts are narrow enough. If this is the case, the new split is assigned to the children n_1 and n_2. Thereby the subtrees rooted at them become meaningless and are removed. Then in line 12 the ImproveSubtree routine is called for the new split. In case no sufficiently narrow split could be found, the old one is reused, provided that it is narrow enough.

If the subtree consists just of the node n, a whole new partition has to be calculated with the call to the Split function in line 2. In case a sufficiently narrow one is found, two new leaves are added to n and labelled with the split. Then for these nodes ImproveSubtree is called. If the ImproveSubtree function is called with the initial rank-decomposition, there is always a new split calculated because it does not exist an old one that could be altered. As already mentioned, the first run always returns a total rank-decomposition. Later this may not always be the case. If Split is not able to find a good partition, or the old one is not good enough and RandomSwap does not find a better one, the node n does not have any children. Hence, ImproveSubtree stops there and returns a partial rank-decomposition.

Splitting a leaf (Split) is done in a greedy way. We start with the partition $(X, Y) = (\emptyset, L^{-1}(n))$ and then greedily move on element at a time from Y to X. To be precise we exchange that $y \in Y$ that leads to a minimal $max\{cutrk(X \cup \{y\}), cutrk(Y \setminus \{y\})\}$. If there are several such y we chose one randomly. In the end, we pick that pair (X, Y) so that $max\{cutrk(X), cutrk(Y)\}$ becomes minimal. Again, there may be more than one partition with equally low width and we have to chose one randomly

There are some constraints for the partition (X, Y), namely X and Y are not allowed to be too small, i.e. $\geq c \cdot \left|L^{-1}(n)\right|$ (and ≥ 1 of course). Later in this work c is referred to as the split factor.

Swapping labels (RandomSwap) is done if the node n has children n_1, n_2. Basically, there is an exchange of elements, i.e. n_1 gets labels from n_2 and n_2 gets labels from n_1. Let X be $L^{-1}(n_1)$ and $Y = L^{-1}(n_2)$. Then the random subsets $X' \subset X$ and $Y' \subset Y$ are moved to the other set. The subsets have some size constraints so that the partition (X, Y) still fulfils the size constraints of the Split function.

Pseudo code implementations of Split and RandomSwap are not shown here for conciseness. Moreover, they are extremely similar to those in [1].

3.2 Results and Discussion

To get real life results, graphs from TreewidthLIB [8] are investigated. Treewidth-LIB is a collection of 710 graphs from many different fields like computanional biology, probabilistic networks, TSP instances and more. Taking these graphs also enables to compare results for boolean-width calculated in [1], tree-width and rank-width on the same graph. We only use graphs that have between 25 and 256 nodes. The algorithm is able to handle much bigger graphs, but only some are tried, mainly because the algorithm that is presented in Section 4 only works on graphs up to that size. Moreover, results for boolean-width were only found for smaller graphs, so a comparison would not be possible.

For many graphs there exist preprocessed versions for tree-width. These are only used if the original graph is too big. That reduction leaves 193 graphs. For 114 of them results for boolean-width exist, for 167 an upper-bound for tree-width is known.

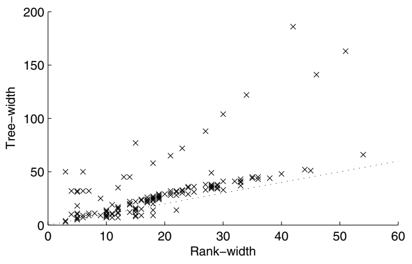
For every graph several runs with different configurations were made and we use the best result that any of this runs could produce. In general, a low split-factor and a high amount of greedy choices lead to better results.

Figure 1 shows a comparison of our results with known upper bounds for tree-width and boolean-width. The dotted line in both Figures marks the equality of both parameters.

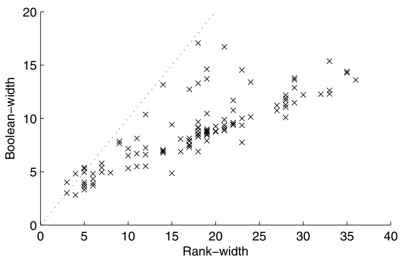
In theory, rank-width can be as high as 2^k on a graph with boolean-width k [12]. While rank-width is sometimes almost equal to or better than boolean-width, in most of the cases boolean-width is significantly lower. The ratio between boolean-width and rank-width is between 1.33 and 0.32 with an average of 0.57.

In [13] a tight bound connecting tree-width and rank-width is established: $rw(G) \leq tw(G) - 1$. This bound is beaten by most of the results the algorithm could produce. On average the known upper bound for $tw(G) - 1$ is 70% higher than our bound for rank-width. We are also able to find rank-decompositions for graphs on which no bounds for tree-width are known.

Some results are shown in Table 1. The first graph `1bkf` is from the field of computational biology. The graph `miles1500` is converted from the stanford graph base [14]. The `myciel6` graph is a Mycielskian graph from the second



(a) Tree-width



(b) Boolean-width

Fig. 1. Comparison of the results with known upper bounds for tree-width and boolean-width

Table 1. Selected results comparing upper bounds for rank-width, boolean-width and tree-width

Graph G	$\|G\|$	$\|\|G\|\|$	$rw(G)$	$boolw(G)$	$tw(G)$
`1bkf`	106	1264	28	11.69	36
`celar08`	458	1655	22	N/A	16
`eil51.tsp`	51	140	7	5.78	9
`fpsol2.i.1`	496	11654	8	N/A	66
`miles1500`	128	5198	15	4.86	77
`mulsol.i.1`	197	3925	3	4.00	50
`myciel6`	95	755	24	13.40	35
`queen7_7`	49	952	12	10.36	35

DIMACS implementation challange [15]. This is also the origin of `fpsol2.i.1` and `mulsol.i.1` which are based on register allocation of real code. A Delaunay triangulation of a travelling salesperson problem is the basis for the `eil51.tsp` graph. The `queen7_7` graph is the graph for a n-queens problem [16]. Finally, `celar08` is a frequency assignment instance [17].

The two examples for a bigger graph, `celar08` and `fpsol2.i.1`, show that the algorithm can also work successfully on bigger graphs.

We used a standard Desktop Computer with a 2.5 GHz Phenom II processor and 16GB of Ram. The algorithm ran until it could not make an improvement for 500 iterations in a row but at most 300 seconds. The actual runtime is in most cases significantly lower. Hvidevold et al. obviously allowed much longer runtimes (c.f. Table 1 in [1]). For `queen8_12` they showed a boolean-width bound of 16.7 in 3055s, whereas we could find a rank-decomposition of width 21 in 69s on that graph. Although they state that they did not aim for "fast benchmark results", this difference is definitely noteworthy.

4 A Lower Bound Algorithm

In order to evaluate the quality of the results in Section 3, one has to know the actual rank-width of the investigated graphs. As it is not known, the algorithm that is presented here tries to calculate it, or to be more precise, it decides if for a given k, there exists a rank-decomposition narrower than k.

The next Section provides an overview on how the algorithm works in principle, before the results are presented in Section 4.2.

4.1 Overview

The algorithm tries to calculate a lower bound by enumerating all possible rank-decompositions for an induced subgraph, then growing the graph and recalculating the new rank-decompositions on the basis of the ones built in the last step. As it is given a maximal width k, which no rank-decomposition may reach, those with a higher width can be excluded. Thereby the search space can be drastically reduced. If at some point no decomposition has a low enough width, the algorithm is stopped because the bound k is a lower bound for the rank-width of the given graph. This is possible because the rank-width of a graph is at most as high as the rank-width of any of its induced subgraphs.

A problem that arises is the very quickly growing number of different rank-decompositions. As there have to be at least $2u$ vertices in a rank-decomposition before any cut possibly reaches the width of the upper bound u, no rank-decomposition can be discarded before they have $2u$ vertices. To drastically reduce the number of rank-decompositions that have to be considered, we only use partial rank-decompositions with exactly two leaves.

Algorithm 3 depicts this in pseudo code. A rank-decomposition that contains only two leaves defines exactly one cut. Therefore, we only store one set per rank-decomposition. When we grow the graph by a vertex v in line 4, two new possible rank-decompositions R_{new} and R are created. One with v in the set and one without it. The set R stays unchanged, for that reason it is important to adapt the Width function in line 7.

Algorithm 3. Main loop

```
Input: set of vertices $V$
 1: $v_1 \in V,\ V \leftarrow V \setminus \{v_1\}$
 2: $R_{init} \leftarrow \{v_1\}$
 3: $\mathcal{R} \leftarrow \{R_{init}\},\ \mathcal{R}' \leftarrow \emptyset$
 4: for all $v \in V$ do
 5:     while $\mathcal{R} \neq \emptyset$ do
 6:         $R \in \mathcal{R}$
 7:         if Width$(R) < k$ then        ▷ Width function applies to current subgraph
 8:             $R_{new} \leftarrow R \cup \{v\}$
 9:             $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{R_{new}, R\}$
10:         end if
11:         $\mathcal{R} \leftarrow \mathcal{R} \setminus \{R\}$
12:     end while
13:     $\mathcal{R} \leftarrow \mathcal{R}',\ \mathcal{R}' \leftarrow \emptyset$
14: end for
```

The graph is grown until either all vertices in V have been inserted or $\mathcal{R}$ is empty. If $\mathcal{R}$ is empty, no rank-decomposition has a width lower than k. Otherwise, the algorithm can not make a decision.

Obviously, there are always decompositions narrower than k in particular all that have a set of size smaller than k. We can show that if a graph has n vertices only those two-leafed rank-decompositions with at least $\frac{1}{3}n$ labels in each leaf represent some unique total rank-decompositions. We can therefore safely ignore them but we have to keep them for the next step of the algorithm, because by adding labels to the right leaf it may fulfil the size constraint.

4.2 Results and Discussion

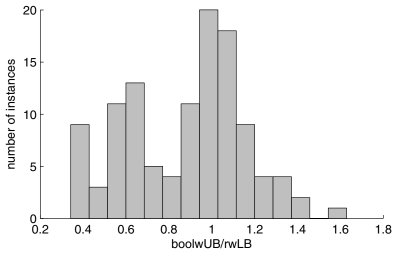
As in Section 3.2, the algorithm was checked against graphs from the treewidth-LIB. For some graphs no upper bound could be found in the time we provided. Contrary to that, we were capable of finding very good lower bounds in only a few minutes on other graphs. In some rare cases, we could even show that the known rank-decomposition is optimal. Surprisingly, we were able to find many lower bounds for rank-width which are above the upper bound for boolean-width.

In total, a lower bound could be found for 179 of the 194 graphs, 10 of which match the upper bound. For 104 graphs there also exists an upper bound for the boolean-width. In 49 cases the upper bound for boolean-width is at most as high as the lower bound for rank-width. This is a remarkable result because it provides a practical evidence that the boolean-width of a graph is in many cases lower than its rank-width.

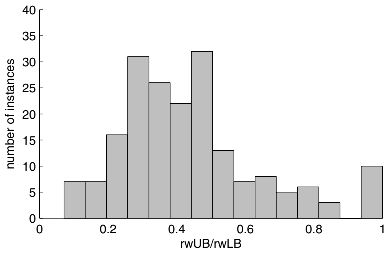
Theoretical results have already indicated this by claiming that $rw \leq 2^{boolw}$. While this bound is known to be tight, it remains unanswered how practically relevant it is especially as on the other hand $boolw \leq \frac{1}{4}rw^2$. Accordingly, it would be possible that for many instances rank-width is the lower width measure. The results in this work allow to say that at least for a part of the investigated graphs boolean-width is lower than rank-width.

Figure 2a shows the relation of boolean-width to the upper bound for rank-with in a histogram. For 104 graphs there exist an upper bound for boolean-width and a lower bound for rank-width. The median of their ratio is 0.95. So for 50% of the graphs we can safely say that its boolean-width is at worst 5% higher than its rank-width. The lower bound algorithm still suffers from limitations. Due to memory restriction we were for example not able to prove a lower bound above 10. Thus, for many graphs the optimal rank-width is significantly higher than the lower bound. Contrariwise, the values for boolean-width are the results of the first attempt to find boolean-decompositions.

The comparison to the upper bounds for rank-width is more difficult, mainly because the calculation of high lower bounds demands more resources than we are able to provide. Nevertheless, we can show the optimality of 10 rank-decompositions. The median of the ratio of lower bound and upper bound is 0.41. Moreover, for 67% of the graphs we can guarantee a 3-approximation. The algorithm by Oum [7] is able to find a 3-approximation for every graph, but there does not exist an implementation until now.



(a) Ratio of boolean-width to the lower bound for rank-width



(b) Ratio of the rank-width upper bound to the lower bound

Fig. 2. Histograms for the lower bound

Table 2. Selected results for the lower bound algorithm, UB: upper bound, LB: lower bound

graph G	$\|G\|$	$\|\|G\|\|$	*rw* UBs	*rw* LBs	*boolw* UBs
`1bx7`	41	195	8	8	4.91
`1kw4`	67	672	22	10	9.39
`BN_12`	90	481	24	8	N/A
`celar03`	200	721	10	5	N/A
`celar06`	100	350	5	5	3.81
`graph01`	100	358	19	5	14.61
`miles750`	128	2113	24	4	N/A
`mulsol.i.2`	188	3885	6	N/A	4.81
`myciel7`	191	2360	54	7	N/A

Besides the mentioned limitations, we are able to calculate many good lower bounds and to find some astonishing results. The results should however not be interpreted as the best achievable. Rather, they merely show a small part of the possibilities.

Table 2 shows a selection of graphs with very different results for the lower bound algorithm. The first two graphs, `1bx7` and `1kw4`, are from the field of computanional biology. `BN_12` is a Bayesian network from evaluation of probabilistic inference systems [18]. Both `celar` graphs as well as `graph01` are frequency assignment instances from the EUCLID CALMA project [17]. Finally, the `mulsol.i.2` graph is a colouring problem generated from a register allocation problem based on real code from the second DIMACS challenge [15].

It was not tried to check if 2 is a lower bound of any graph, as this is only the case if a graph is distance hereditary [19]. This could be checked much easier and does most likely not apply to any of the available graphs. Thus, we can safely assume a lower bound of 2.

5 Conclusion and Outlook

A heuristic algorithm for the calculation of rank-decompositions was developed and tested against multiple graphs of TreewidthLIB. The algorithm is able to find rank-decompositions in a fast way. Unfortunately, their width is only low for a few graphs and in most cases significantly worse than the known bounds for boolean-width. As it was not clear if this is caused by the high rank-width of these graphs or by a bad result of the algorithm, a second algorithm was developed, which is able to decide if a graph has a rank-decomposition of width lower than k. Runtime and memory usage increase significantly with k, therefore tight bounds could not be found in most cases. However, we were often able to push it near or even above the known boolean-width. This evidence suggests that boolean-width is on graphs from real life application in fact a better, i.e. lower parameter.

Both algorithms use a rather simple approach and are presumably the first practical algorithms for upper and lower bounds of rank-width. Nevertheless, the results are to some extent exceptional, e.g. a rank-decomposition of width 8 on the large `fpsol2.i.1` graph or rank-width exactly 5 on the graph `celar06` .

The results presented in this work encourage further research on boolean-width. It would be of particular interest to find a fast and memory-efficient way to calculate $|UN(X)|$ (see equation 1). Both algorithms which we developed could then be adapted for boolean-width. Concerning the heuristic algorithm, even better values would be reachable, assuming that changing the parameters of the algorithm has a similar effect as for rank-width. Furthermore, a possibility to calculate lower bounds would be at hand.

Apart from that, this results also suggests further theoretical work as boolean-width seems to outperform the known width-measures. A strong theoretical background and access to good decompositions will enable to practically solve many hard problems on graphs.

References

1. Hvidevold, E.M., Sharmin, S., Telle, J.A., Vatshelle, M.: Finding Good Decompositions for Dynamic Programming on Dense Graphs. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 219–231. Springer, Heidelberg (2012)
2. Courcelle, B.: The monadic second-order logic of graphs i. recognizable sets of finite graphs. Information and Computation, 12–75 (1990)
3. Courcelle, B., Makowsky, J., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique width. Theory of Computing Systems 33, 125–150 (1999)
4. Langer, A., Reidl, F., Rossmanith, P., Sikdar, S.: Recent progress in practical aspects of mso model-checking (in preparation, 2012)
5. Bodlaender, H.L., Koster, A.M.: Treewidth computations i. upper bounds. Information and Computation 208(3), 259–275 (2010)
6. Hliněny, P., Oum, S.I., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. Computer Journal, 10–1093 (2007)

7. Oum, S.I.: Approximating rank-width and clique-width quickly. ACM Trans. Algorithms 5(1), 10:1–10:20 (2008)
8. Bodlaender, H., van den Broek, J.W.: Treewidthlib: A benchmark for algorithms for treewidth and related graph problems (2004), http://www.cs.uu.nl/research/projects/treewidthlib/
9. Robertson, N., Seymour, P.: Graph minors. iii. planar tree-width. Journal of Combinatorial Theory, Series B 36(1), 49–64 (1984)
10. Robertson, N., Seymour, P.: Graph minors. x. obstructions to tree-decomposition. Journal of Combinatorial Theory, Series B 52(2), 153–190 (1991)
11. Oum, S.I., Seymour, P.: Approximating clique-width and branch-width. J. Comb. Theory Ser. B 96, 514–528 (2006)
12. Bui-Xuan, B.M., Telle, J.A., Vatshelle, M.: Boolean-width of graphs. Theoretical Computer Science 412(39), 5187–5204 (2011)
13. Oum, S.I.: Rank-width is less than or equal to branch-width. Journal of Graph Theory 57(3), 239–244 (2008)
14. Knuth, D.E.: The Stanford GraphBase: a platform for combinatorial computing. ACM, New York (1993)
15. Johnson, D.J., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993. American Mathematical Society, Boston (1996)
16. Bell, J., Stevens, B.: A survey of known results and research areas for n-queens. Discrete Mathematics 309(1), 1–31 (2009)
17. Rlfap, E., Eindhoven, T.U., Group, R.: Euclid calma radio link frequency assignment project technical annex t-2.3.3: Local search (1995)
18. Bilmes, J.: Uai 2006 inference evaluation results. Technical report, University of Washington, Seattle (2006)
19. Oum, S.I.: Rank-width and vertex-minors. J. Comb. Theory Ser. B 95(1), 79–100 (2005)

Determinacy in Stochastic Games with Unbounded Payoff Functions[⋆]

Tomáš Brázdil[⋆⋆], Antonín Kučera[⋆⋆], and Petr Novotný[⋆⋆]

Faculty of Informatics, Masaryk University
{xbrazdil,kucera,xnovot18}@fi.muni.cz

Abstract. We consider infinite-state turn-based stochastic games of two players, $\Box$ and $\Diamond$, who aim at maximizing and minimizing the expected total reward accumulated along a run, respectively. Since the total accumulated reward is unbounded, the determinacy of such games cannot be deduced directly from Martin's determinacy result for Blackwell games. Nevertheless, we show that these games *are* determined both for unrestricted (i.e., history-dependent and randomized) strategies and deterministic strategies, and the equilibrium value is the same. Further, we show that these games are generally *not* determined for memoryless strategies. Then, we consider a subclass of $\Diamond$-*finitely-branching* games and show that they are determined for all of the considered strategy types, where the equilibrium value is always the same. We also examine the existence and type of (ε-)optimal strategies for both players.

1 Introduction

Turn-based stochastic games of two players are a standard model of discrete systems that exhibit both non-deterministic and randomized choice. One player (called $\Box$ or Max in this paper) corresponds to the controller who wishes to achieve/maximize some desirable property of the system, and the other player (called $\Diamond$ or Min) models the environment which aims at spoiling the property. Randomized choice is used to model events such as system failures, bit-flips, or coin-tossing in randomized algorithms.

Technically, a turn-based stochastic game (SG) is defined as a directed graph where every vertex is either stochastic or belongs to one of the two players. Further, there is a fixed probability distribution over the outgoing transitions of every stochastic vertex. A *play* of the game is initiated by putting a token on some vertex. Then, the token is moved from vertex to vertex by the players or randomly. A *strategy* specifies how a player should play. In general, a strategy may depend on the sequence of vertices visited so far (we say that the strategy is *history-dependent (H)*), and it may specify a probability distribution over the outgoing transitions of the currently visited vertex rather than a single outgoing transtion (we say that the strategy is *randomized (R)*). Strategies that do not depend on the history of a play are called *memoryless (M)*, and strategies that do not randomize (i.e., select a single outgoing transition) are called *determinisctic (D)*. Thus, we obtain the MD, MR, HD, and HR strategy classes, where HR are unrestricted strategies and MD are the most restricted memoryless deterministic strategies.

[⋆] The full version of this paper can be found at http://arxiv.org/abs/1208.1639

[⋆⋆] The authors are supported by the Czech Science Foundation, grant No. P202/12/G061.

A *game objective* is usually specified by a *payoff function* which assigns some real value to every run (infinite path) in the game graph. The aim of Player $\Box$ is to *maximize* the expected payoff, while Player $\Diamond$ aims at *minimizing* it. It has been shown in [22] that for *bounded* and *Borel* payoff functions, Martin's determinacy result for Blackwell games [23] implies that

$$\sup_{\sigma \in \mathrm{HR}_\Box} \inf_{\pi \in \mathrm{HR}_\Diamond} \mathbb{E}_v^{\sigma,\pi}[\mathit{Payoff}] \quad = \quad \inf_{\pi \in \mathrm{HR}_\Diamond} \sup_{\sigma \in \mathrm{HR}_\Box} \mathbb{E}_v^{\sigma,\pi}[\mathit{Payoff}] \tag{1}$$

where $\mathrm{HR}_\Box$ and $\mathrm{HR}_\Diamond$ are the classes of HR strategies for Player $\Box$ and Player $\Diamond$, respectively. Hence, every vertex v has a *HR-value* $\mathrm{Val}_{\mathrm{HR}}(v)$ specified by (1). A HR strategy is *optimal* if it achieves the outcome $\mathrm{Val}_{\mathrm{HR}}(v)$ or better against every strategy of the other player. In general, optimal strategies are not guaranteed to exist, but (1) implies that both players have *ε-optimal* HR strategies for every $\varepsilon > 0$ (see Section 2 for precise definitions).

The determinacy results of [23,22] cannot be applied to *unbounded* payoff functions, i.e., these results do not imply that (1) holds if *Payoff* is unbounded, and they do not say anything about the existence of a value for restricted strategy classes such as MD or MR. In the context of performance analysis and controller synthesis, these questions rise naturally; in some cases, the players cannot randomize or remember the history of a play, and some of the studied payoff functions are not bounded. In this paper, we study these issues for the *total accumulated reward* payoff function and *infinite-state* games.

The total accumulated reward payoff function, denoted by *Acc*, is defined as follows. Assume that every vertex v is assigned a fixed non-negative reward $r(v)$. Then *Acc* assigns to every run the sum of rewards of all vertices visited along the run. Obviously, *Acc* is unbounded in general, and may even take the ∞ value. A special case of a total accumulated reward is the *termination time*, where all vertices are assigned reward 1, except for terminal vertices that are assigned reward 0 (we also assume that the only outgoing transition of every terminal vertex t is a self-loop on t). Then, $\mathbb{E}_v^{\sigma,\pi}[\mathit{Acc}]$ corresponds to the expected termination time under the strategies σ, π. Another special (and perhaps the simplest) case of a total accumulated reward is *reachability*, where the target vertices are assigned reward 1 and the other vertices have zero reward (here we assume that every target vertex has a single outgoing transition to a special state s with zero reward, where $s \to s$ is the only outgoing transition of s). Although the reachability payoff is bounded, some of our negative results about the total accumulated reward hold even for reachability (see below).

The reason for considering infinite-state games is that many recent works study various algorithmic problems for games over classical automata-theoretic models, such as pushdown automata [15,16,17,14,9,8], lossy channel systems [3,2], one-counter automata [7,5,6], or multicounter automata [18,11,10,21,13,4], which are finitely representable, but for which the underlying game graph is infinite and sometimes even infinitely-branching (see, e.g., [11,10,21]). Since the properties of finite-state games do *not* carry over to infinite-state games in general (see, e.g., [20]), the above issues need to be revisited and clarified explicitly, which is the main goal of this paper.

Our Contribution: We consider general infinite-state games, which may contain vertices with infinitely many outgoing transitions, and $\Diamond$-finitely-branching games, where every vertex controlled by player $\Diamond$ has finitely many outgoing transitions, with the total accumulated reward objective. For *general* games, we show the following:

- Every vertex has both a HR and a HD-value, and these values are equal[1].
- There is a vertex v of a game G with a reachability objective such that v has neither MD nor MR-value. Further, the game G has only one vertex (belonging to Player $\Diamond$) with infinitely many outgoing transitions.

It follows from previous works (see, e.g., [8,20]) that optimal strategies in general games may not exist, and even if they do exist, they may require infinite memory. Interestingly, we observe that an optimal strategy for Player $\Box$ (if it exists) may also require randomization in some cases.

For $\Diamond$*-finitely-branching* games, we prove the following results:

- Every vertex has a HR, HD, MR, and MD-value, and all of these values are equal.
- Player $\Diamond$ has an optimal MD strategy in every vertex.

It follows from the previous works that Player $\Box$ may not have an optimal strategy and even if he has one, it may require infinite memory. Let us note that in finite-state games, both players have optimal MD strategies (see, e.g., [19]).

Our results are obtained by generalizing the arguments for reachability objectives presented in [8], but there are also some new observations based on original ideas and new counterexamples. In particular, this applies to the existence of a HD-value and the non-existence of MD and MR-values in general games.

Due to the space constraints, most proofs are omitted. They can be found in the full version of this paper [12].

2 Preliminaries

In this paper, the sets of all positive integers, non-negative integers, rational numbers, real numbers, and non-negative real numbers are denoted by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{R}^{\geq 0}$, respectively. We also use $\mathbb{R}^{\geq 0}_{\infty}$ to denote the set $\mathbb{R}^{\geq 0} \cup \{\infty\}$, where ∞ is treated according to the standard conventions. For all $c \in \mathbb{R}^{\geq 0}_{\infty}$ and $\varepsilon \in [0, \infty)$, we define the *lower* and *upper* ε-approximation of c, denoted by $c \ominus \varepsilon$ and $c \oplus \varepsilon$, respectively, as follows:

$$\begin{aligned} c \oplus \varepsilon &= c + \varepsilon && \text{for all } c \in \mathbb{R}^{\geq 0}_{\infty} \text{ and } \varepsilon \in [0, \infty), \\ c \ominus \varepsilon &= c - \varepsilon && \text{for all } c \in \mathbb{R}^{\geq 0} \text{ and } \varepsilon \in [0, \infty), \\ \infty \ominus \varepsilon &= 1/\varepsilon && \text{for all } \varepsilon \in (0, \infty), \\ \infty \ominus 0 &= \infty\,. \end{aligned}$$

Given a set V, the elements of $(\mathbb{R}^{\geq 0}_{\infty})^V$ are written as vectors $\boldsymbol{x}, \boldsymbol{y}, \ldots$, where $\boldsymbol{x}_v$ denotes the v-component of $\boldsymbol{x}$ for every $v \in V$. The standard component-wise ordering on $(\mathbb{R}^{\geq 0}_{\infty})^V$ is denoted by $\sqsubseteq$.

[1] For a given strategy type T (such as MD or MR), we say that a vertex v has a *T value* if $\sup_{\sigma \in T_\Box} \inf_{\pi \in T_\Diamond} \mathbb{E}_v^{\sigma,\pi}[\mathit{Payoff}] = \inf_{\pi \in T_\Diamond} \sup_{\sigma \in T_\Box} \mathbb{E}_v^{\sigma,\pi}[\mathit{Payoff}]$, where $T_\Box$ and $T_\Diamond$ are the classes of all T strategies for Player $\Box$ and Player $\Diamond$, respectively.

For every finite or countably infinite set M, a binary relation $\rightarrow \subseteq M \times M$ is *total* if for every $m \in M$ there is some $n \in M$ such that $m \rightarrow n$. A *finite path* in $\mathcal{M} = (M, \rightarrow)$ is a finite sequence $w = m_0, \ldots, m_k$ such that $m_i \rightarrow m_{i+1}$ for every i, where $0 \leq i < k$. The *length* of w, i.e., the number of transitions performed along w, is denoted by $|w|$. A *run* in $\mathcal{M}$ is an infinite sequence $\omega = m_0, m_1, \ldots$ every finite prefix of which is a path. We also use $\omega(i)$ to denote the element m_i of ω. Given $m, n \in M$, we say that n is *reachable* from m, written $m \rightarrow^* n$, if there is a finite path from m to n. The sets of all finite paths and all runs in $\mathcal{M}$ are denoted by $\mathit{Fpath}(\mathcal{M})$ and $\mathit{Run}(\mathcal{M})$, respectively. For every finite path w, we use $\mathit{Run}(\mathcal{M}, w)$ and $\mathit{Fpath}(\mathcal{M}, w)$ to denote the set of all runs and finite paths, respectively, prefixed by w. If $\mathcal{M}$ is clear from the context, we write just Run, $\mathit{Run}(w)$, Fpath and $\mathit{Fpath}(w)$ instead of $\mathit{Run}(\mathcal{M})$, $\mathit{Run}(\mathcal{M}, w)$, $\mathit{Fpath}(\mathcal{M})$ and $\mathit{Fpath}(\mathcal{M}, w)$, respectively.

Now we recall basic notions of probability theory. Let A be a finite or countably infinite set. A *probability distribution* on A is a function $f : A \rightarrow \mathbb{R}^{\geq 0}$ such that $\sum_{a \in A} f(a) = 1$. A distribution f is *positive* if $f(a) > 0$ for every $a \in A$, *Dirac* if $f(a) = 1$ for some $a \in A$, and *uniform* if A is finite and $f(a) = \frac{1}{|A|}$ for every $a \in A$. A *σ-field* over a set X is a set $\mathcal{F} \subseteq 2^X$ that includes X and is closed under complement and countable union. A *measurable space* is a pair $(X, \mathcal{F})$ where X is a set called *sample space* and $\mathcal{F}$ is a σ-field over X. A *probability measure* over a measurable space $(X, \mathcal{F})$ is a function $\mathcal{P} : \mathcal{F} \rightarrow \mathbb{R}^{\geq 0}$ such that, for each countable collection $\{X_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\mathcal{P}(\bigcup_{i \in I} X_i) = \sum_{i \in I} \mathcal{P}(X_i)$, and moreover $\mathcal{P}(X) = 1$. A *probability space* is a triple $(X, \mathcal{F}, \mathcal{P})$ where $(X, \mathcal{F})$ is a measurable space and $\mathcal{P}$ is a probability measure over $(X, \mathcal{F})$.

Definition 1. *A* stochastic game *is a tuple $G = (V, \rightarrow, (V_\Box, V_\Diamond, V_\bigcirc), \mathit{Prob})$ where V is a finite or countably infinite set of* vertices, *$\rightarrow \subseteq V \times V$ is a total* transition relation, *$(V_\Box, V_\Diamond, V_\bigcirc)$ is a partition of V, and Prob is a* probability assignment *which to each $v \in V_\bigcirc$ assigns a positive probability distribution on the set of its outgoing transitions. We say that G is* $\Diamond$-finitely-branching *if for each $v \in V_\Diamond$ there are only finitely many $u \in V$ such that $v \rightarrow u$.*

Strategies. A stochastic game G is played by two players, $\Box$ and $\Diamond$, who select the moves in the vertices of $V_\Box$ and $V_\Diamond$, respectively. Let $\odot \in \{\Box, \Diamond\}$. A *strategy* for Player $\odot$ in G is a function which to each finite path in G ending in a vertex $v \in V_\odot$ assigns a probability distribution on the set of outgoing transitions of v. We say that a strategy τ is *memoryless (M)* if $\tau(w)$ depends just on the last vertex of w, and *deterministic (D)* if it returns a Dirac distribution for every argument. Strategies that are not necessarily memoryless are called *history-dependent (H)*, and strategies that are not necessarily deterministic are called *randomized (R)*. Thus, we obtain the MD, MR, HD, and HR *strategy types*. The set of all strategies for Player $\odot$ of type T in a game G is denoted by $T_\odot^G$, or just by $T_\odot$ if G is understood (for example, $\mathrm{MR}_\Box$ denotes the set of all MR strategies for Player $\Box$).

Every pair of strategies $(\sigma, \pi) \in \mathrm{HR}_\Box \times \mathrm{HR}_\Diamond$ and an initial vertex v determine a unique probability space $(\mathit{Run}(v), \mathcal{F}, \mathcal{P}_v^{\sigma,\pi})$, where $\mathcal{F}$ is the smallest σ-field over $\mathit{Run}(v)$ containing all the sets $\mathit{Run}(w)$ such that w starts with v, and $\mathcal{P}_v^{\sigma,\pi}$ is the unique probability measure such that for every finite path $w = v_0, \ldots, v_k$ initiated in v we have that $\mathcal{P}_v^{\sigma,\pi}(Run(w)) = \Pi_{i=0}^{k-1} x_i$, where x_i is the probability of $v_i \to v_{i+1}$ assigned either by $\sigma(v_0, \ldots, v_i)$, $\pi(v_0, \ldots, v_i)$, or $Prob(v_i)$, depending on whether v_i belongs to $V_\Box$, $V_\Diamond$, or $V_\bigcirc$, respectively (in the case when $k = 0$, i.e., $w = v$, we put $\mathcal{P}_v^{\sigma,\pi}(Run(w)) = 1$).

Determinacy, Optimal Strategies. In this paper, we consider games with the *total accumulated reward* objective and *reachability* objective, where the latter is understood as a restricted form of the former (see below).

Let $r : V \to \mathbb{R}^{\geq 0}$ be a *reward function*, and $Acc : Run \to \mathbb{R}^{\geq 0}_{\infty}$ a function which to every run ω assigns the *total accumulated reward* $Acc(\omega) = \sum_{i=0}^{\infty} r(\omega(i))$. Let T be a strategy type. We say that a vertex $v \in V$ *has a T-value* in G if

$$\sup_{\sigma \in T_\Box} \inf_{\pi \in T_\Diamond} \mathbb{E}_v^{\sigma,\pi}[Acc] \quad = \quad \inf_{\pi \in T_\Diamond} \sup_{\sigma \in T_\Box} \mathbb{E}_v^{\sigma,\pi}[Acc],$$

where $\mathbb{E}_v^{\sigma,\pi}[Acc]$ denotes the expected value of Acc in $(Run(v), \mathcal{F}, \mathcal{P}_v^{\sigma,\pi})$. If v has a T-value, then $\mathrm{Val}_T(v, r, G)$ (or just $\mathrm{Val}_T(v)$ if G and r are clear from the context) denotes the *T-value of v* defined by this equality.

Let $\mathcal{G}$ be a class of games. If every vertex of every $G \in \mathcal{G}$ has a T-value for every reward function, we say that $\mathcal{G}$ is *T-determined*. Note that Acc is generally not bounded, and therefore we cannot directly apply the results of [23,22] to conclude that the class of all games is HR-determined. Further, these results do not say anything about the determinacy for the other strategy types even for bounded objective functions.

If a given vertex v has a T-value, we can define the notion of ε-optimal T strategy for both players.

Definition 2. *Let v be a vertex which has a T-value, and let $\varepsilon \geq 0$. We say that*

- $\sigma \in T_\Box$ *is ε-T-*optimal *in v if* $\mathbb{E}_v^{\sigma,\pi}[Acc] \geq \mathrm{Val}_T(v) \ominus \varepsilon$ *for all* $\pi \in T_\Diamond$*;*
- $\pi \in T_\Diamond$ *is ε-T-*optimal *in v if* $\mathbb{E}_v^{\sigma,\pi}[Acc] \leq \mathrm{Val}_T(v) \oplus \varepsilon$ *for all* $\sigma \in T_\Box$.

*A 0-T-optimal strategy is called T-*optimal.

In this paper we also consider *reachability* objectives, which can be seen as a restricted form of the total accumulated reward objectives introduced above. A "standard" definition of the reachability payoff function looks as follows: We fix a set $R \subseteq V$ of *target* vertices, and define a function $Reach : Run \to \{0, 1\}$ which to every run assigns either 1 or 0 depending on whether or not the run visits a target vertex. Note that $\mathbb{E}_v^{\sigma,\pi}[Reach]$ is the *probability* of visiting a target vertex in the corresponding play of G. Obviously, if we assign reward 1 to the target vertices and 0 to the others, and replace all outgoing transitions of target vertices with a single transition leading to a fresh stochastic vertex u with reward 0 and only one transition $u \to u$, then $\mathbb{E}_v^{\sigma,\pi}[Reach]$ in the original game is equal to $\mathbb{E}_v^{\sigma,\pi}[Acc]$ in the modified game. Further, if the original game was $\Diamond$-finitely-branching or finite, then so is the modified game. Therefore, all "positive" results about the total accumulated reward objective (e.g., determinacy, existence of T-optimal strategies, etc.) achieved in this paper carry over to the reachability objective, and all "negative" results about reachability carry over to the total accumulated reward.

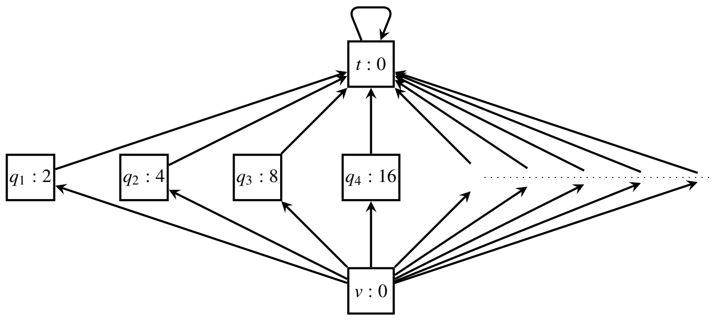


Fig. 1. Player □ has an MR-optimal strategy in v, but no HD-optimal strategy in v. All vertices are labelled by pairs of the form *vertex name*:*reward*.

3 Results

Our main results about the determinacy of general stochastic games with the total accumulated reward payoff function are summarized in the following theorem:

Theorem 3. *Let $\mathcal{G}$ be the class of all games. Then*

a) $\mathcal{G}$ is both HR-determined and HD-determined. Further, for every vertex v of every $G \in \mathcal{G}$ and every reward function r we have that $\mathrm{Val}_{\mathrm{HR}}(v) = \mathrm{Val}_{\mathrm{HD}}(v)$.
b) $\mathcal{G}$ is neither MD-determined nor MR-determined, and these results hold even for reachability objectives.

An optimal strategy for Player □ does not necessarily exist, even if G is a game with a reachability payoff function such that $V_\Diamond = \emptyset$ and every vertex of $V_\Box$ has at most two outgoing transitions (see, e.g., [8,20]). In fact, it suffices to consider the vertex v of Fig. 2 where the depicted game is modified by replacing the vertex u with a stochastic vertex u', where $u' \to u'$ is the only outgoing transition of u', and u' is the only target vertex (note that all vertices in the first two rows become unreachable and can be safely deleted). Clearly, $\mathrm{Val}_{\mathrm{HR}}(v) = 1$, but Player □ has no optimal strategy.

Similarly, an optimal strategy for Player ◇ may not exist even if $V_\Box = \emptyset$ [8,20]. To see this, consider the vertex u of Fig. 2, where t is the only target vertex and the depicted game is modified by redirecting the only outgoing transition of p back to u (this makes all vertices in the last two rows unreachable). We have that $\mathrm{Val}_{\mathrm{HR}}(u) = 0$, but Player ◇ has no optimal strategy.

One may be also tempted to think that if Player □ (or Player ◇) has *some* optimal strategy, then he also has an optimal MD strategy. However, optimal strategies generally require *infinite memory* even for reachability objectives (this holds for both players). Since the corresponding counterexamples are not completely trivial, we refer to [20] for details. Interestingly, an optimal strategy for Player □ may also require *randomization*. Consider the vertex v of Fig. 1. Let $\sigma^* \in \mathrm{MR}_\Box$ be a strategy selecting $v \to q_n$ with probability $1/2^n$. Since $V_\Diamond = \emptyset$, we have that $\inf_{\pi \in \mathrm{HR}_\Diamond} \mathbb{E}_v^{\sigma^*,\pi}[Acc] = \infty = \mathrm{Val}_{\mathrm{HR}}(v)$. However, for every $\sigma \in \mathrm{HD}_\Box$ we have that $\inf_{\pi \in \mathrm{HR}_\Diamond} \mathbb{E}_v^{\sigma,\pi}[Acc] < \infty$.

For $\Diamond$-finitely-branching games, the situation is somewhat different, as our second main theorem reveals.

Theorem 4. *Let $\mathcal{G}$ be the class of all $\Diamond$-finitely-branching games. Then $\mathcal{G}$ is HR-determined, HD-determined, MR-determined, and MD-determined, and for every vertex v of every $G \in \mathcal{G}$ and every reward function r we have that*

$$\mathrm{Val}_{\mathrm{HR}}(v) = \mathrm{Val}_{\mathrm{HD}}(v) = \mathrm{Val}_{\mathrm{MR}}(v) = \mathrm{Val}_{\mathrm{MD}}(v)\,.$$

Further, for every $G \in \mathcal{G}$ there exists an MD strategy for Player $\Diamond$ which is optimal in every vertex of G.

An optimal strategy for Player $\Box$ may not exist in $\Diamond$-finitely-branching games, and even if it does exist, it may require infinite memory [20].

Theorems 3 and 4 are proven by a sequence of lemmas presented below. For the rest of this section, we fix a stochastic game $G = (V, \rightarrow, (V_\Box, V_\Diamond, V_\bigcirc), \mathit{Prob})$ and a reward function $r\colon V \rightarrow \mathbb{R}^{\geq 0}$. We start with the first part of Theorem 3 (a), i.e., we show that every vertex has a HR-value. This is achieved by defining a suitable Bellman operator L and proving that the least fixed-point of L is the tuple of all HR-values. More precisely, let $L\colon (\mathbb{R}^{\geq 0}_{\infty})^V \rightarrow (\mathbb{R}^{\geq 0}_{\infty})^V$, where $\boldsymbol{y} = L(\boldsymbol{x})$ is defined as follows:

$$\boldsymbol{y}_v = \begin{cases} r(v) + \sup_{v \rightarrow v'} \boldsymbol{x}_{v'} & \text{if } v \in V_\Box \\ r(v) + \inf_{v \rightarrow v'} \boldsymbol{x}_{v'} & \text{if } v \in V_\Diamond \\ r(v) + \sum_{v \rightarrow v'} \boldsymbol{x}_{v'} \cdot \mathit{Prob}(v)(v, v') & \text{if } v \in V_\bigcirc. \end{cases}$$

A proof of the following lemma can be found in the full version of this paper. Some parts of this proof are subtle, and we also need to make several observations that are useful for proving the other results.

Lemma 5. *The operator L has the least fixed point $\boldsymbol{K}$ (w.r.t. $\sqsubseteq$) and for every $v \in V$ we have that*

$$\boldsymbol{K}_v \quad = \quad \sup_{\sigma \in \mathrm{HR}_\Box} \inf_{\pi \in \mathrm{HR}_\Diamond} \mathbb{E}_v^{\sigma,\pi}[\mathit{Acc}] \quad = \quad \inf_{\pi \in \mathrm{HR}_\Diamond} \sup_{\sigma \in \mathrm{HR}_\Box} \mathbb{E}_v^{\sigma,\pi}[\mathit{Acc}] \quad = \quad \mathrm{Val}_{\mathrm{HR}}(v)\,.$$

Moreover, for every $\varepsilon > 0$ there is $\pi_\varepsilon \in \mathrm{HD}_\Diamond$ such that for every $v \in V$ we have that $\sup_{\sigma \in \mathrm{HR}_\Box} \mathbb{E}_v^{\sigma,\pi_\varepsilon} \leq \mathrm{Val}_{\mathrm{HR}}(v) \oplus \varepsilon$.

To complete our proof of Theorem 3 (a), we need to show the existence of a HD-value in every vertex, and demonstrate that HR and HD values are equal. Due to Lemma 5, for every $\varepsilon > 0$ there is $\pi_\varepsilon \in \mathrm{HD}_\Diamond$ such that π_ε is ε-HR-optimal in every vertex. Hence, it suffices to show the same for Player $\Box$. The following lemma is proved in the full version.

Lemma 6. *For every $\varepsilon > 0$, there is $\sigma_\varepsilon \in \mathrm{HD}_\Box$ such that σ_ε is ε-*HR*-optimal in every vertex.*

The next lemma proves Item (b) of Theorem 3.

Lemma 7. *Consider the vertex v of the game shown in Fig. 2, where t is the only target vertex and all probability distributions assigned to stochastic states are uniform. Then*

(a) $\sup_{\sigma\in\mathrm{MD}_\Box}\inf_{\pi\in\mathrm{MD}_\Diamond}\mathbb{E}_v^{\sigma,\pi}[\mathit{Reach}] = \sup_{\sigma\in\mathrm{MR}_\Box}\inf_{\pi\in\mathrm{MR}_\Diamond}\mathbb{E}_v^{\sigma,\pi}[\mathit{Reach}] = 0$;
(b) $\inf_{\pi\in\mathrm{MD}_\Diamond}\sup_{\sigma\in\mathrm{MD}_\Box}\mathbb{E}_v^{\sigma,\pi}[\mathit{Reach}] = \inf_{\pi\in\mathrm{MR}_\Diamond}\sup_{\sigma\in\mathrm{MR}_\Box}\mathbb{E}_v^{\sigma,\pi}[\mathit{Reach}] = 1$.

Proof. We start by proving item (a) for MD strategies. Let $\sigma^* \in \mathrm{MD}_\Box$. We show that $\inf_{\pi\in\mathrm{MD}_\Diamond}\mathbb{E}_v^{\sigma^*,\pi}[\mathit{Reach}] = 0$. Let us fix an arbitrarily small $\varepsilon > 0$. We show that there is a suitable $\pi^* \in \mathrm{MD}_\Diamond$ such that $\mathbb{E}_v^{\sigma^*,\pi^*}[\mathit{Reach}] \leq \varepsilon$. If the probability of reaching the vertex u from v under the strategy σ^* is at most ε, we are done. Otherwise, let p_s be the probability of visiting the vertex s from v under the strategy σ *without* passing through the vertex u. Note that $p_s > 0$ and p_s does not depend on the strategy chosen by Player $\Diamond$. The strategy π^* selects a suitable successor of u such that the probability p_t of visiting the vertex t from u without passing through the vertex v satisfies $p_t/p_s < \varepsilon$ (note that p_t can be arbitrarily small but positive). Then

$$\mathbb{E}_v^{\sigma^*,\pi^*}[\mathit{Reach}] \quad\leq\quad \sum_{i=1}^{\infty}(1-p_s)^i p_t \quad=\quad \frac{(1-p_s)p_t}{p_s} \quad\leq\quad \varepsilon\,.$$

For MR strategies, the argument is the same.

Item (b) is proven similarly. We show that for all $\pi^* \in \mathrm{MD}_\Diamond$ and $0 < \varepsilon < 1$ there exists a suitable $\sigma^* \in \mathrm{MD}_\Box$ such that $\mathbb{E}_v^{\sigma^*,\pi^*}[\mathit{Reach}] \geq 1-\varepsilon$. Let p_t be the probability of visiting t from u without passing through the vertex v under the strategy π^*. We choose the strategy σ^* so that the probability p_s of visiting the vertex s from v without passing through the vertex u satisfies $p_s/p_t < \varepsilon$. Note that almost all runs initiated in v eventually visit either s or t under (σ^*,π^*). Since the probability of visiting s is bounded by ε (the computation is similar to the one of item (a)), we obtain $\mathbb{E}_v^{\sigma^*,\pi^*}[\mathit{Reach}] \geq 1-\varepsilon$. For MR strategies, the proof is almost the same. □

We continue by proving Theorem 4. This theorem follows immediately from Lemma 5 and the following proposition:

Proposition 8. *If G is $\Diamond$-finitely-branching, then*

1. *for all $v \in V$ and $\varepsilon > 0$, there is $\sigma_\varepsilon \in \mathrm{MD}_\Box$ such that σ_ε is ε-HR-optimal in v;*
2. *there is $\pi \in \mathrm{MD}_\Diamond$ such that π is HR-optimal in every vertex.*

As an immediate corollary to Proposition 8, we obtain the following result:

Corollary 9. *If G is $\Diamond$-finitely-branching, $V_\Box$ is finite, and every vertex of $V_\Box$ has finitely many successors, then there is $\sigma \in \mathrm{MD}_\Box$ such that σ is HR-optimal in every vertex.*

Proof. Due to Proposition 8, for every vertex v and every $\varepsilon > 0$, there is $\sigma_\varepsilon \in \mathrm{MD}_\Box$ such that σ_ε is ε-HR-optimal in v. Since $V_\Box$ is finite and every vertex of $V_\Box$ has only finitely many successors, there are only finitely many MD strategies for Player $\Box$. Hence, there is a MD strategy σ that is ε-HR-optimal in v for infinitely many ε from the set $\{1, 1/2, 1/4, \ldots\}$. Such a strategy is clearly HR-optimal in v. Note that σ is HR-optimal in every vertex which can be reached from v under σ and some strategy π for Player $\Diamond$. For the remaining vertices, we can repeat the argument, and thus eventually produce a MD strategy that is HR-optimal in every vertex. □

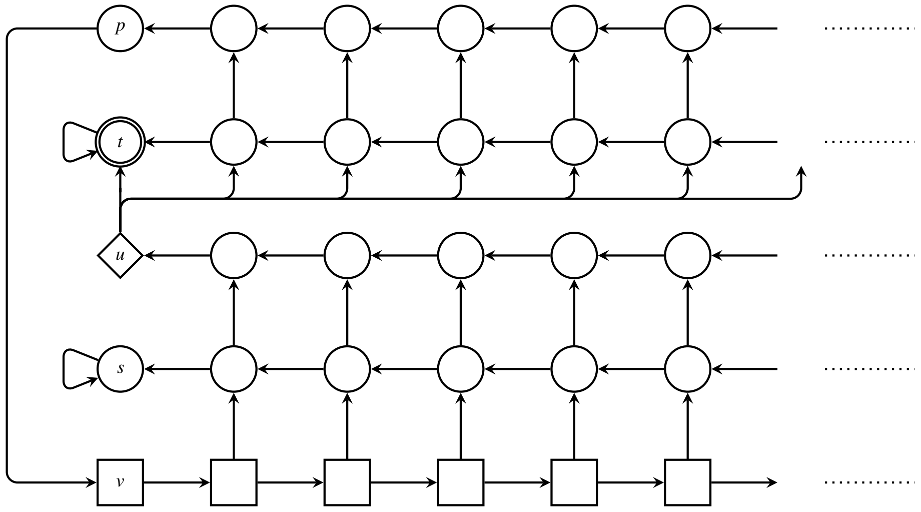


Fig. 2. A game whose vertex v has neither MD-value nor MR-value

Hence, if all non-stochastic vertices have finitely many successors and $V_\Box$ is finite, then both players have HR-optimal MD strategies. This can be seen as a (tight) generalization of the corresponding result for finite-state games [19].

The rest of this section is devoted to a proof of Proposition 8. We start with Item 1. The strategy σ_ε is constructed by employing discounting. Assume, w.l.o.g., that rewards are bounded by 1 (if they are not, we may split every state v with a reward $r(v)$ into a sequence of $\lceil r(v) \rceil$ states, each with the reward $r(v)/\lceil r(v) \rceil$). Given $\lambda \in (0,1)$, define $Acc^\lambda : Run \to \mathbb{R}^{\geq 0}$ to be a function which to every run ω assigns $Acc^\lambda(\omega) = \sum_{i=0}^{\infty} \lambda^i \cdot r(\omega(i))$.

Lemma 10. *For λ sufficiently close to one we have that*

$$\sup_{\sigma \in \mathrm{HR}_\Box} \inf_{\pi \in \mathrm{HR}_\Diamond} \mathbb{E}_v^{\sigma,\pi}(Acc^\lambda) \quad \geq \quad \mathrm{Val}_{\mathrm{HR}}(v) \ominus \frac{\varepsilon}{2}\,.$$

Proof. We show that for every $\varepsilon > 0$ there is $n \geq 0$ such that the expected reward that Player $\Box$ may accumulate up to n steps is ε-close to $\mathrm{Val}_{\mathrm{HR}}(v)$ no matter what Player $\Diamond$ is doing. Formally, define $Acc_k : Run \to \mathbb{R}^{\geq 0}$ to be a function which to every run ω assigns $Acc_k(\omega) = \sum_{i=0}^{k} r(\omega(i))$. The following lemma is proved in the full version of this paper.

Lemma 11. *If G is $\Diamond$-finitely-branching, then for every $v \in V$ there is $n \in \mathbb{N}$ such that*

$$\sup_{\sigma \in \mathrm{HR}_\Box} \inf_{\pi \in \mathrm{HR}_\Diamond} \mathbb{E}_v^{\sigma,\pi}(Acc_n) \quad > \quad \mathrm{Val}_{\mathrm{HR}}(v) \ominus \frac{\varepsilon}{4}\,.$$

Clearly, if λ is close to one, then for every run ω we have that

$$Acc^\lambda(\omega) \quad \geq \quad Acc_n(\omega) - \frac{\varepsilon}{4}\,.$$

Thus,

$$\sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma,\pi}(Acc^\lambda) \quad\geq\quad \sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma,\pi}(Acc_n)-\frac{\varepsilon}{4} \quad\geq\quad \mathrm{Val}_{\mathrm{HR}}(v)\ominus\frac{\varepsilon}{2}\,.$$

This proves Lemma 10. □

So, it suffices to find an MD strategy σ_ε satisfying

$$\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma_\varepsilon,\pi}(Acc^\lambda) \quad\geq\quad \sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma,\pi}(Acc^\lambda)-\frac{\varepsilon}{2}\,.$$

We define such a strategy as follows. Let us fix some $\ell\in\mathbb{N}$ satisfying

$$\frac{\lambda^\ell}{1-\lambda}\cdot\max_{v\in V}r(v) \quad<\quad \frac{\varepsilon}{8}\,.$$

Intuitively, the discounted reward accumulated after ℓ steps can be at most $\frac{\varepsilon}{8}$. In a given vertex $v\in V_\Box$, the strategy σ_ε chooses a fixed successor vertex u satisfying

$$\sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_u^{\sigma,\pi}(Acc^\lambda) \quad\geq\quad \sup_{v\to u'}\sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_{u'}^{\sigma,\pi}(Acc^\lambda)-\frac{\varepsilon}{\ell\cdot 4}$$

Now we show that

$$\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma_\varepsilon,\pi}(Acc^\lambda) \quad\geq\quad \sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma,\pi}(Acc^\lambda)-\frac{\varepsilon}{2}\,.$$

which finishes the proof of Item 1 of Proposition 8.

For every $k\in\mathbb{N}$ we denote by σ_k a strategy for Player □ defined as follows: For the first k steps the strategy behaves similarly to σ_ε, i.e., chooses, in each state $v\in V_\Box$, a next state u satisfying

$$\sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_u^{\sigma,\pi}(Acc^\lambda) \quad\geq\quad \sup_{v\to u'}\sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_{u'}^{\sigma,\pi}(Acc^\lambda)-\frac{\varepsilon}{k\cdot 4}\,.$$

From $k{+}1$-st step on, say in a state u, the strategy follows some strategy ζ satisfying

$$\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_u^{\zeta,\pi}(Acc^\lambda) \quad\geq\quad \sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_u^{\sigma,\pi}(Acc^\lambda)-\frac{\varepsilon}{8}\,.$$

A simple induction reveals that σ_k satisfies

$$\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma_k,\pi}(Acc^\lambda) \quad\geq\quad \sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma,\pi}(Acc^\lambda)-\frac{3\varepsilon}{8}\,. \tag{2}$$

(Intuitively, the error of each of the first k steps is at most $\frac{\varepsilon}{k\cdot 4}$ and thus the total error of the first k steps is at most $k\cdot\frac{\varepsilon}{k\cdot 4}=\frac{\varepsilon}{4}$. The rest has the error at most $\frac{\varepsilon}{8}$ and thus the total error is at most $\frac{3\varepsilon}{8}$.)

We consider $k=\ell$ (recall that $\frac{\lambda^\ell}{1-\lambda}\cdot\max_{v\in V}r(v)<\frac{\varepsilon}{8}$). Then

$$\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma_\varepsilon,\pi}(Acc^\lambda) \quad\geq\quad \inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma_k,\pi}(Acc^\lambda)-\frac{\varepsilon}{8} \quad\geq\quad \sup_{\sigma\in\mathrm{HR}_\Box}\inf_{\pi\in\mathrm{HR}_\Diamond}\mathbb{E}_v^{\sigma,\pi}(Acc^\lambda)-\frac{\varepsilon}{2}\,.$$

Here the first equality follows from the fact that σ_k behaves similarly to σ_ε on the first $k = \ell$ steps and the discounted reward accumulated after k steps is at most $\frac{\varepsilon}{8}$. The second inequality follows from Equation (2).

It remains to prove Item 2 of Proposition 8. The MD strategy π can be easily constructed as follows: In every state $v \in V_\Diamond$, the strategy π chooses a successor u minimizing $\mathrm{Val}_{\mathrm{HR}}(u)$ among all successors of v. We show in the full version that this is indeed an optimal strategy.

4 Conclusions

We have considered infinite-state stochastic games with the total accumulated reward objective, and clarified the determinacy questions for the HR, HD, MR, and MD strategy types. Our results are almost complete. One natural question which remains open is whether Player $\Box$ needs memory to play ε-HR-optimally in general games (it follows from the previous works, e.g., [8,20], that ε-HR-optimal strategies for Player $\Diamond$ require infinite memory in general).

References

1. Proceedings of FST&TCS 2010, Leibniz International Proceedings in Informatics, vol. 8. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2010)
2. Abdulla, P.A., Ben Henda, N., de Alfaro, L., Mayr, R., Sandberg, S.: Stochastic Games with Lossy Channels. In: Amadio, R.M. (ed.) FoSSaCS 2008. LNCS, vol. 4962, pp. 35–49. Springer, Heidelberg (2008)
3. Baier, C., Bertrand, N., Schnoebelen, P.: On Computing Fixpoints in Well-Structured Regular Model Checking, with Applications to Lossy Channel Systems. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 347–361. Springer, Heidelberg (2006)
4. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite Runs in Weighted Timed Automata with Energy Constraints. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 33–47. Springer, Heidelberg (2008)
5. Brázdil, T., Brožek, V., Etessami, K.: One-counter stochastic games. In: Proceedings of FST&TCS 2010 [1], pp. 108–119
6. Brázdil, T., Brožek, V., Etessami, K., Kučera, A.: Approximating the Termination Value of One-Counter MDPs and Stochastic Games. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 332–343. Springer, Heidelberg (2011)
7. Brázdil, T., Brožek, V., Etessami, K., Kučera, A., Wojtczak, D.: One-counter Markov decision processes. In: Proceedings of SODA 2010, pp. 863–874. SIAM (2010)
8. Brázdil, T., Brožek, V., Forejt, V., Kučera, A.: Reachability in recursive Markov decision processes. Information and Computation 206(5), 520–537 (2008)
9. Brázdil, T., Brožek, V., Kučera, A., Obdržálek, J.: Qualitative reachability in stochastic BPA games. Information and Computation 208(7), 772–796 (2010)
10. Brázdil, T., Chatterjee, K., Kučera, A., Novotný, P.: Efficient Controller Synthesis for Consumption Games with Multiple Resource Types. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 23–38. Springer, Heidelberg (2012)
11. Brázdil, T., Jančar, P., Kučera, A.: Reachability Games on Extended Vector Addition Systems with States. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 478–489. Springer, Heidelberg (2010)

12. Brázdil, T., Kučera, A., Novotný, P.: Determinacy in stochastic games with unbounded payoff functions. CoRR abs/1208.1639 (2012)
13. Chatterjee, K., Doyen, L., Henzinger, T., Raskin, J.F.: Generalized mean-payoff and energy games. In: Proceedings of FST&TCS 2010 [1], pp. 505–516
14. Etessami, K., Wojtczak, D., Yannakakis, M.: Recursive Stochastic Games with Positive Rewards. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 711–723. Springer, Heidelberg (2008)
15. Etessami, K., Yannakakis, M.: Recursive Markov Decision Processes and Recursive Stochastic Games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 891–903. Springer, Heidelberg (2005)
16. Etessami, K., Yannakakis, M.: Efficient Qualitative Analysis of Classes of Recursive Markov Decision Processes and Simple Stochastic Games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 634–645. Springer, Heidelberg (2006)
17. Etessami, K., Yannakakis, M.: Recursive Concurrent Stochastic Games. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 324–335. Springer, Heidelberg (2006)
18. Fahrenberg, U., Juhl, L., Larsen, K.G., Srba, J.: Energy Games in Multiweighted Automata. In: Cerone, A., Pihlajasaari, P. (eds.) ICTAC 2011. LNCS, vol. 6916, pp. 95–115. Springer, Heidelberg (2011)
19. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer (1996)
20. Kučera, A.: Turn-based stochastic games. In: Apt, K.R., Grädel, E. (eds.). Lectures in Game Theory for Computer Scientists, pp. 146–184. Cambridge University Press (2011)
21. Kučera, A.: Playing Games with Counter Automata. In: Finkel, A., Leroux, J., Potapov, I. (eds.) RP 2012. LNCS, vol. 7550, pp. 29–41. Springer, Heidelberg (2012)
22. Maitra, A., Sudderth, W.: Finitely additive stochastic games with Borel measurable payoffs. International Journal of Game Theory 27, 257–267 (1998)
23. Martin, D.: The determinacy of Blackwell games. Journal of Symbolic Logic 63(4), 1565–1581 (1998)

Strategy Complexity of Finite-Horizon Markov Decision Processes and Simple Stochastic Games*

Krishnendu Chatterjee[1] and Rasmus Ibsen-Jensen[2]

[1] IST Austria
krish.chat@ist.ac.at
[2] Department of Computer Science, Aarhus University, Denmark
rij@cs.au.dk

Abstract. Markov decision processes (MDPs) and simple stochastic games (SSGs) provide a rich mathematical framework to study many important problems related to probabilistic systems. MDPs and SSGs with finite-horizon objectives, where the goal is to maximize the probability to reach a target state in a given finite time, is a classical and well-studied problem. In this work we consider the strategy complexity of finite-horizon MDPs and SSGs. We show that for all $\epsilon > 0$, the natural class of counter-based strategies require at most $\log\log(\frac{1}{\epsilon}) + n + 1$ memory states, and memory of size $\Omega(\log\log(\frac{1}{\epsilon}) + n)$ is required, for ϵ-optimality, where n is the number of states of the MDP (resp. SSG). Thus our bounds are asymptotically optimal. We then study the periodic property of optimal strategies, and show a sub-exponential lower bound on the period for optimal strategies.

1 Introduction

Markov Decision Process and Simple Stochastic Games. The class of *Markov decision processes (MDPs)* is a classical model for probabilistic systems that exhibit both stochastic and and deterministic behavior [4]. MDPs have been widely used to model and solve control problems for stochastic systems [3]: there, non-determinism represents the freedom of the controller to choose a control action, while the probabilistic component of the behavior describes the system response to control actions. *Simple stochastic games (SSGs)* enrich MDPs by allowing two types of non-determinism (angelic and demonic non-determinism) along with stochastic behavior [1]. MDPs and SSGs provide a rich mathematical framework to study many important problems related to probabilistic systems.

Finite-Horizon Objective. One classical problem widely studied for MDPs and SSGs is the *finite-horizon objective*. In a finite-horizon objective, a finite time horizon T is

* Work of the second author supported by the Sino-Danish Center for the Theory of Interactive Computation, funded by the Danish National Research Foundation and the National Science Foundation of China (under the grant 61061130540). The second author acknowledge support from the Center for research in the Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council. The first author was supported by FWF Grant No P 23499-N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

given and the goal of the player is to maximize the payoff within the time horizon T in MDPs (in SSGs against all strategies of the opponent). The complexity of MDPs and SSGs with finite-horizon objectives have been well studied, with book chapters dedicated to them [3,7]. The complexity results basically show that iterating the Bellman equation for T steps yield the desired result [3,7]. While the computational complexity have been well-studied, perhaps surprisingly the strategy complexity has not received great attention. In this work we consider several problems related to the strategy complexity of MDPs and SSGs with finite-horizon objectives, where the objective is to reach a target state within a finite time horizon T.

Our Contribution. In this work we consider the memory requirement for ϵ-optimal strategies, for $\epsilon > 0$, and a periodic property of optimal strategies in finite-horizon MDPs and SSGs. A strategy is an ϵ-optimal strategy, for $\epsilon > 0$, if the strategy ensures within ϵ of the optimal value against all strategies of the opponent. For finite-horizon objectives, the natural class of strategies are counter-based strategies, which has a counter to count the number of time steps. Our first contribution is to establish asymptotically optimal memory bounds for ϵ-optimal counter-based strategies, for $\epsilon > 0$, in finite-horizon MDPs and SSGs. We show that ϵ-optimal counter-based strategies require at most memory of size $\log\log(\frac{1}{\epsilon}) + n + 1$ and memory of size $\Omega(\log\log(\frac{1}{\epsilon}) + n)$ is required, where n is the size of the state space. Thus our bounds are asymptotically optimal. The upper bound holds for SSGs and the lower bound is for MDPs. We then consider the periodic (or regularity) property of optimal strategies. The period of a strategy is the number P such that the strategy repeats within every P steps (i.e., it is periodic with time step P). We show a sub-exponential lower bound on the period of optimal strategies for MDPs with finite-horizon objectives, by presenting a family of MDPs with n states where all optimal strategies are periodic and the period is $2^{\Omega(\sqrt{n\cdot\log(n)})}$.

Organization of the Paper. The paper is organized as follows: In Section 2 we present all the relevant definitions related to stochastic games and strategies. In Section 3 we show that $\Theta(n + \log\log\epsilon^{-1})$ number of bits are necessary and sufficient for ϵ-optimal counter-based strategies, for all $\epsilon > 0$, in both finite-horizon MDPs and SSGs. In Section 4 we show that there are finite-horizon MDPs where all optimal strategies are periodic and have a period of $2^{\Omega(\sqrt{n\log n})}$. Detailed proofs available at http://arxiv.org/abs/1209.3617.

2 Definitions

The class of *infinite-horizon simple stochastic games* (SSGs) consists of two player, zero-sum, turn-based games, played on a (multi-)graph. The class was first defined by Condon [1]. Below we define SSGs, the finite-horizon version, and the important subclass of MDPs.

SSGs, finite-horizon SSGs, and MDPs. An SSG $G = (S_1, S_2, S_R, \perp, (A_s)_{s\in S_1\cup S_2\cup S_R}, s_0)$ consists of a terminal state $\perp$ and three sets of disjoint non-terminal states, S_1 (max state), S_2 (min states), S_R (coin toss states). We will use S to denote the union, i.e., $S = S_1 \cup S_2 \cup S_R$. For each state $s \in S$, let A_s be a (multi-)set of *outgoing arcs of* s. We will use $A = \bigcup_s A_s$ to denote the (multi-)set of all arcs. Each state $s \in S$ has two outgoing arcs. If a is a arc, then $d(a) \in S \cup \{\perp\}$ is the *destination* of a. There is also a designated start state $s_0 \in S$. The class of *finite-horizon simple stochastic games* (FSSGs) also consists of two player, zero-sum, turn-based games, played on a (multi-)graph. An FSSG (G, T) consists of an SSG G and a finite time limit (or horizon) $T \geq 0$. Let G be an SSG and $T \geq 0$, then we will write the FSSG (G, T) as G^T. Given an SSG G (resp. FSSG G^T), for a state s, we denote by G_s (resp. G_s^T) the same game as G (resp. G^T), except that s is the start state. The class of *infinite (resp. finite) horizon Markov decision processes* (MDPs and FMDPs respectively) is the subclass of SSGs (resp. FSSGs) where $S_2 = \emptyset$.

Plays and objectives of the players. An SSG G is *played* as follows. A pebble is moved on to s_0. For $i \in \{1, 2\}$, whenever the pebble is moved on to a state s in S_i, then Player i chooses some arc $a \in A_s$ and moves the pebble to $d(a)$. Whenever the pebble is moved on to a state s in S_R, then an $a \in A_s$ is chosen uniformly at random and the pebble moves to $d(a)$. If the pebble is moved on to $\perp$, then the game is over. For all $T \geq 0$ the FSSG G^T is played like G, except that the pebble can be moved at most $T + 1$ times. The *objective* of both SSGs and FSSGs is for Player 1 to maximize the probability that the pebble is moved on to $\perp$ (eventually in SSGs and with in $T+1$ time steps in FSSGs). The objective of Player 2 is to minimize this probability.

Strategies. Let S^* be the set of finite sequences of states. For all T, let $S^{\leq T} \subset S^*$ be the set of sequences of states, which have length at most T. A *strategy σ_i for Player i* in an SSG is a map from $S^* \times S_i$ into A, such that for all $w \in S^*$ and $s \in S$ we have $\sigma_i(w \cdot s) \in A_s$. Similarly, a *strategy σ_i for Player i* in an FSSG G^T is a map from $S^{\leq T} \times S_i$ into A, such that for all $w \in S^{\leq T}$ and $s \in S$ we have $\sigma_i(w \cdot s) \in A_s$. In all cases we denote by Π_i the set of all strategies for Player i. If $S_i = \emptyset$, we will let $\emptyset$ denote the corresponding strategy set. Below we define some special classes of strategies.

Memory-based, counter-based and Markov strategies. Let $M = \{0, 1\}^*$ be the set of possible *memories*. A *memory-based strategy σ_i for Player i* consists of a pair (σ_u, σ_a), where (i) σ_u, the memory-update function, is a map from $M \times S$ into M; and (ii) σ_a, the next-action function, is a map from $M \times S_i$ into A, such that for all $m \in M$ and $s \in S_i$ we have $\sigma_a(m, s) \in A_s$. A *counter-based strategy* is a special case of memory-based strategies, where for all $m \in M$ and $s, s' \in S$ we have $\sigma_u(m, s) = \sigma_u(m, s')$. That is the memory can only contain a counter of some type. We will therefore write $\sigma_u(m, s)$ as $\sigma_u(m)$ for all m, s and any counter-based strategy σ. A *Markov strategy σ_i for Player i* is a special case of strategies where

$$\forall p, p' \in S^{\leq T} : |p| = |p'| \wedge p_{|p|} = p'_{|p'|} \in S_i \Rightarrow \sigma(p', p'_{|p'|}) = \sigma(p, p_{|p|}).$$

That is, a Markov strategy only depends on the length of the history and the current state. Let Π_i' be the set of all Markov strategies for Player i.

Following a strategy. For a strategy, σ_i, for Player i we will say that Player i *follows* σ_i if for all n given the sequence of states $(p_i)_{i \leq n}$ the pebble has been on until move n and that $p_n \in S_i$, then Player i chooses $\sigma((p_i)_{i \leq n}, p_n)$. For a memory-based strategy for Player i σ_i, we will say that Player i *follows* σ_i if for all n given the sequence of states

$(p_i)_{i \le n}$ the pebble has been on until move n, that $p_n \in S_i$ and that $m^i = \sigma_u(m^{i-1}, p_i)$ and that $m^0 = \emptyset$, then Player i chooses $\sigma_a(m^n, p_n)$.

Space required by a memory-based strategy. The *space usages of a memory-based strategy* is the logarithm of the number of distinct states generated by the strategy at any point, if the player follows that strategy. A memory-based strategy is *memoryless* if there is only one memory used by the strategy. For any FSSG G^T with n states it is clear that the set of strategies is a subset of memory-based strategies that uses memory at most $T \log n$, since for any strategy σ we can construct a memory-based strategy σ' by using the memory for the sequence of states and then choose the same action as σ would with that sequence of states. Hence we will also talk about ϵ-optimal memory-based strategies. Also note that for any FSSG G^T it is clear that the set of Markov strategies is a subset of the set of counter-based strategies that uses space at most $\log T$.

Period of a counter-based strategy. We will distinguish between two kinds of memories for a counter-based strategy σ. One kind is only used once (the initial phase) and the other kind is used arbitrarily many times (the periodic phase). Let $m^0 = \emptyset$ and $m^i = \sigma_u(m^{i-1})$. Then if $m^i = m^j$ for some $i < j$, we also have that $m^{i+c} = m^{j+c}$ and $m^i = m^{i+c(j-i)}$. Hence if a memory is used twice, it will be reused again. We will let the number of memories that are only used once be N and the number of memories used more than once be p, which we will call the period. The number N is mainly important for ϵ-optimal strategies and period is mainly important for optimal strategies.

Probability measure and values. A pair of strategies (σ_1, σ_2), one for each player (in either an SSG or an FSSG), defines a probability that the pebble is eventually moved to $\perp$. Let the probability be denoted as P^{σ_1,σ_2}. For all SSGs G (resp. FSSGs G^T) it follows from the results of Everett [2] that $\sup_{\sigma_1 \in \Pi_1'} \inf_{\sigma_2 \in \Pi_2} P^{\sigma_1,\sigma_2} = \inf_{\sigma_2 \in \Pi_2'} \sup_{\sigma_1 \in \Pi_1} P^{\sigma_1,\sigma_2}$. We will call this common value as the *value* of G (resp. G^T) and denote it $\mathrm{val}(G)$ (resp. $\mathrm{val}(G^T)$).

ϵ-optimal and optimal strategies. For all $\epsilon \ge 0$, we will say that a strategy σ_1 is *ϵ-optimal for Player 1* if $\inf_{\sigma_2 \in \Pi_2} P^{\sigma_1,\sigma_2} + \epsilon \ge \sup_{\sigma_1' \in \Pi_1'} \inf_{\sigma_2 \in \Pi_2} P^{\sigma_1',\sigma_2}$. Similarly, a strategy σ_2 is *ϵ-optimal for Player 2* if $\sup_{\sigma_1 \in \Pi_1} P^{\sigma_1,\sigma_2} - \epsilon \le \inf_{\sigma_2' \in \Pi_2'} \sup_{\sigma_1 \in \Pi_1} P^{\sigma_1,\sigma_2'}$. A strategy σ is *optimal for Player i* if it is 0-optimal. Condon [1] showed that *there exist optimal memoryless strategies* for any SSG G that are also optimal for G_s for all $s \in S$. This also implies that there are optimal Markov strategies for FSSGs that are also optimal for G_s for all $s \in S$.

3 Bounds on ϵ-Optimal Counter-Based Strategies

We will first show an upper bound on size of the memory used by a counter-based strategy for playing ϵ-optimal in time limited games. The upper bound on memory size is by application of a result from Ibsen-Jensen and Miltersen [5]. The idea of the proof is that if we play an optimal strategy of G in G^T for sufficiently high T, then the value we get approaches the value of G.

Theorem 1. (Upper bound) *For all FSSGs G^T with n states and $\epsilon > 0$, there is an ϵ-optimal counter-based strategy for both players such that memory size is at most* $\log \log \epsilon^{-1} + n + 1$

Proof. Since there is an optimal Markov strategy, there is a counter-based strategy, which uses memory at most $\log T$. As shown by Ibsen-Jensen and Miltersen [5] for any game G^T, if the horizon is greater than $2\log \epsilon^{-1}2^n$, the value of G^T approximates the value of G with in ϵ. It is clear that the value of all states are the same in an infinite-horizon game if either player is forced to play an optimal strategy. Hence, if $T \geq 2\log \epsilon^{-1}2^n$ and either player plays an optimal strategy of G in G^T, then the value of all states are within ϵ of the value of the game. But there are optimal memoryless strategies in G as shown by Condon [1]. Therefore we have that in the worst case $T < 2\log \epsilon^{-1}2^n$. Since $\log T$ is an upper bound, $\log\log \epsilon^{-1} + n + 1$ is also an upper bound and hence the result. □

We will now lower bound the size of the memory needed for a counter-based strategy to be ϵ-optimal. Our lower bound will be divided into two parts. The first part will show that $\log\log \epsilon^-1$ is a lower bound on the memory required even for some MDPs with constantly many states. The second part will show that even for fixed ϵ, an ϵ-optimal counter-based strategy will need to use a memory of size $O(n)$. Both lower bounds will show explicit MDPs with the required properties. See Figure 1 and Figure 2 respectively.

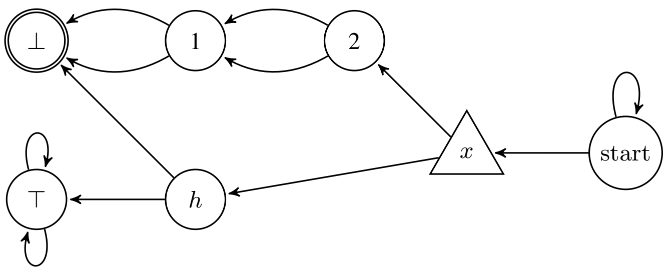


Fig. 1. An MDP G, such that for all $\epsilon > 0$ there is a T, such that all ϵ-optimal memory-based strategies for G^T require memory size of at least $\Omega(\log\log \epsilon^{-1})$. Circle vertices are the coin toss states. The triangle vertex is the max state. The vertex $\perp$ is the terminal state.

MDP for the lower bound of $\log\log \epsilon^{-1}$. Our first lower bound shows that in the MDP M (Figure 1) all ϵ-optimal memory-based strategies require at least $\log \epsilon^{-1}$ distinct memory states, i.e., the size of memory is at least $\log\log \epsilon^{-1}$. The MDP M is defined as follows. There is one state x in S_1, the rest are in S_R.

- The state $\top \in S_R$ has $A_\top = \{(\top,\top),(\top,\top)\}$.
- The state $h \in S_R$ has $A_h = \{(h,\top),(h,\perp)\}$.
- The state $1 \in S_R$ has $A_1 = \{(1,\perp),(1,\perp)\}$.
- The state $2 \in S_R$ has $A_2 = \{(2,1),(2,1)\}$.
- The state $x \in S_1$ has $A_m = \{(x,2),(x,h)\}$.
- The state start$\in S_R$ has $A_{\text{start}} = \{(\text{start},\text{start}),(\text{start},x)\}$.

Lemma 1. *All ϵ-optimal memory-based strategies in M^T, for $T = \log \epsilon^{-1} - 1$, require at least $\log \epsilon^{-1} - 2$ distinct states of memory, i.e., the size of memory is at least $\log \log \epsilon^{-1}$.*

Proof. We will first show the proof for counter-based strategies. At the end we will then extend it to memory-based strategies.

It is clear that $\mathrm{val}(M_x^2) = \frac{1}{2}$ and for all $T > 2$ we have $\mathrm{val}(M_x^T) = 1$. If Player 1 chooses (x, h) in M_x^2, then he gains $\frac{1}{2}$, otherwise, if he chooses $(x, 2)$, then he gains 0. Also for all $T > 2$, if Player 1 chooses $(x, 2)$ in M_x^T, then he gains 1, otherwise, if he chooses (x, h), then he gains $\frac{1}{2}$.

In M_{start} we end up at x after precisely $k \geq 2$ moves of the pebble with probability 2^{-k+1}. Therefore, by the preceding any optimal memory-based strategy σ must be able to find out if T minus the length of the history is greater than 2 from the memory.

Let $\epsilon > 0$ be given. For simplicity we will assume that $\epsilon = 2^{-k}$ for some $k > 0$. Let $c = \log \epsilon^{-1}$. Assume now that there is a counter-based strategy $\sigma = (\sigma_u, \sigma_a)$ that uses $c-3$ states of memory in M_{start}^{c-1}. The pebble ends up at m after $c-3$ moves with probability $2^{-(c-3)+1} = 4\epsilon$. Let the sequences of memories until then be $m^0, m^1, \ldots, m^{c-3}$. Since σ was ϵ-optimal we must have that $\sigma(m^{c-3}, x) = (x, h)$. On the other hand for all $i < c - 3$ we must also have that $\sigma(m^i, x) = (x, 2)$. Therefore m^{c-3} differs from m^i for $i < c - 3$. Now assume that $m^i = m^j$ for $i < j$ and $i, j < c - 3$. But then $\sigma_u(m^i) = \sigma_u(m^j)$ and hence $m^{i+1} = m^{j+1}$ and then by repeating this argument we have that $m^k = m^{c-3}$ for $k < c - 3$. Therefore m^i differs from m^j for $i \neq j$ and $i, j \leq c - 3$ and hence we need at least $c - 2$ different memory states.

For general memory-based strategies the proof remains the same. This is because we can note that if the pebble ends up at x after $c - 3$ moves, we have that $m^0 = \emptyset$ and $m^i = \sigma_u(m^{i-1}, \text{start})$ for $1 \leq i \leq c - 3$ and hence they must all differ by the same argument as before. □

For our second lower bound we will use an infinite family of MDPs $H = \{H(1), H(2), \ldots, H(i), \ldots\}$, such that $H(i)$ contains $2i + 4$ states, one of which is a max state, and all ϵ-optimal counter-based strategies require space at least $i - 4$, for some fixed ϵ.

Family of MDPs for the lower bound of n. The MDP $H(i)$ is defined as follows. There is one state x in S_1, the rest are in S_R.

- The state $\top \in S_R$ has $A_\top = \{(\top, \top), (\top, \top)\}$.
- The state $h \in S_R$ has $A_h = \{(h, \top), (h, \bot)\}$.
- The state $1 \in S_R$ has $A_1 = \{(1, \bot), (1, i)\}$.
- For $j \in \{2, \ldots, i\}$, the state $j \in S_R$ has $A_j = \{(j, i), (j, j-1)\}$.
- The state $x \in S_1$ has $A_m = \{(x, i), (x, h)\}$.
- The state $1^* \in S_R$ has $A_{1^*} = \{(1^*, i^*), (1^*, x)\}$.
- For $j \in \{2, \ldots, i\}$, the state $j^* \in S_R$ has $A_{j^*} = \{(j^*, i^*), (j^*, (j-1)^*)\}$.

There is a illustration of $H(4)$ in Figure 2.

Let i be some number. It is clear that $\mathrm{val}(H(i)_x^2) = \frac{1}{2}$. It is also easy to see that $\mathrm{val}(H(i)_i) = 1$, but that the time to reach $\bot$ from i is quite long. Hence, one can deduce that there must be a k (k depends on i) such that for all $k' \geq k$ it is an optimal strategy in $H(i)_x^{k'}$ to choose (x, i) and for all $2 \leq k'' < k$ it is an optimal strategy in $H(i)_x^{k''}$ to choose (x, h). In case there are multiple such numbers, let k be the smallest. The number $k-1$ is then the smallest number of moves of the pebble to reach $\perp$ from i, such that that occurs with probability $\geq \frac{1}{2}$ (to simplify the proofs we will assume equality).

Let p^t be the probability for the pebble to reach x from i^* in t or less moves (note that this is also the probability to reach $\perp$ in t moves or less from i). It is clear that p^t is equal to the probability that a sequence of t fair coin tosses contains i consecutive tails. This is known to be exactly $1 - F_{t+2}^{(i)}/2^t$, where $F_{t+2}^{(i)}$ is the $(t+2)$'nd Fibonacci i-step number, i.e. the number given by the linear homogeneous recurrence $F_c^{(i)} = \sum_{j=1}^{i} F_{c-j}^{(i)}$ and the boundary conditions $F_c^{(i)} = 0$, for $c \leq 0$, $F_1^{(i)} = F_2^{(i)} = 1$ (this fact is also mentioned in Ibsen-Jensen and Miltersen [5]). We establish several properties of p^t, $F_a^{(i)}$ and k (omitted due to lack of space) and establish the following key result that shows k is exponential in i.

Lemma 2. *For all i, we have that $k \geq 2^{i-2} + i$.*

Proof. We will first show that $p^a \leq p^{a-1} + 2^{-i}$. We can divide the event that there are i consecutive tails into two possibilities out of t fair coin tosses. Either the first i coin tosses were tails or there are i consecutive tails in the last $t-1$ coin tosses (or both). The first case happens with probability 2^{-i} and the last with probability p^{a-1}. We can then apply union bounds and get that $p^a \leq p^{a-1} + 2^{-i}$. Clearly we have that $p^{i-1} = 0$ and that p^a is increasing in a. But we also have that

$$p^k \leq 2^{i-2}2^{-i} + p^{k-2^{i-2}} \Rightarrow \frac{1}{2} \leq \frac{1}{4} + p^{k-2^{i-2}} \Rightarrow \frac{1}{4} \leq p^{k-2^{i-2}},$$

which means that $k > 2^{i-2} + i - 1$. □

Lemma 3. *There is an ϵ such that for all $i \geq 12$, there is a time-bound T such that all ϵ-optimal counter-based strategies for $H(i)^T$ require memory size at least $i-5$.*

The proof basically goes as follows: The pebble starts at i^* with $2k+1$ moves remaining. First we show that there is a super-constant probability for the pebble to reach x using somewhere between $\frac{k}{5}$ and $\frac{4k}{5}$ moves. In that case there is at least $\frac{6k}{5}+1$ moves left. We then show that there is some number $p > \frac{1}{2}$ independent of i such that the probability to reach $\perp$ from i in $\frac{6k}{5}$ is more than p. Secondly we show that there is a super-constant probability for the pebble to reach x using somewhere between $\frac{6k}{5}$ and $\frac{9k}{5}$ moves. In that case there is at most $\frac{4k}{5}+1$ moves left. We then show that there is some number $q < \frac{1}{2}$ independent of i such that the probability to reach $\perp$ from i in $\frac{4k}{5}$ is less than q. We can then pick ϵ such that any ϵ-optimal strategy must distinguish between plays that used between $\frac{k}{5}$ and $\frac{4k}{5}$ moves to reach x from i^* and plays that used between $\frac{6k}{5}$ and $\frac{9k}{5}$ moves to reach x from i^*. We then show that that requires at least $O(k)$ distinct states of memory, and the result then follows from k being exponential in i, by Lemma 2. Thus from from Lemma 1 and Lemma 3 we have the following result.

Theorem 2. (Lower bound) *For all sufficiently small $\epsilon > 0$ and all $n \geq 5$, there is a FMDP with n states, where all ϵ-optimal counter-based strategies require memory size at least $\Omega(\log\log\epsilon^{-1} + n)$.*

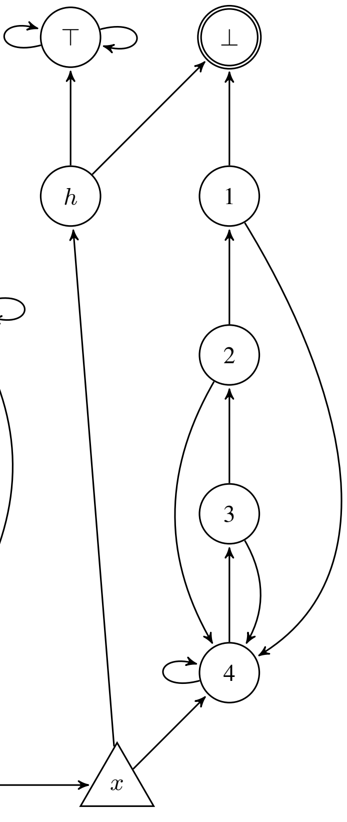


Fig. 2. The MDP $H(4)$. It is the fourth member of a family that will show that there exist FSSGs where, for a fixed ϵ, all ϵ-optimal counter-based strategies require memory size to be at least $\Omega(i)$. Circle vertices are the coin toss states. The triangle vertex is the max state. The vertex $\perp$ is the terminal state.

4 A Lower Bound on the Period of Optimal Strategies in MDPs

In this section we will show that there exist FMDPs G, with n states, such that all optimal strategies can be implemented using a counter-based strategy, and the period is greater than $2^{\Omega(\sqrt{n \log n})}$. We will create such FMDPs in two steps. First we will construct a family, such that the i'th member requires that one state uses one action every $\Theta(i)$ steps and in all other steps uses the other action. There is an illustration of a member of that family in Figure 3. Afterwards we will play many such games in parallel, which will ensure that a large period is needed for all optimal strategies. There is an illustration of such a game in Figure 4.

Let G_p, $p \in \{2, 3, \dots\}$ be the following FMDP, with $2p-1$ coin toss states and one max state. The coin toss states are divided into the sets $\{1^*, 2^*, \dots, (p-1)^*\}$ and $\{1, 2, \dots, p\}$. To simplify the following description let state 0^* denote the $\perp$ terminal state. A description of G (an illustration of G_5 in Figure 3) is then as follows:

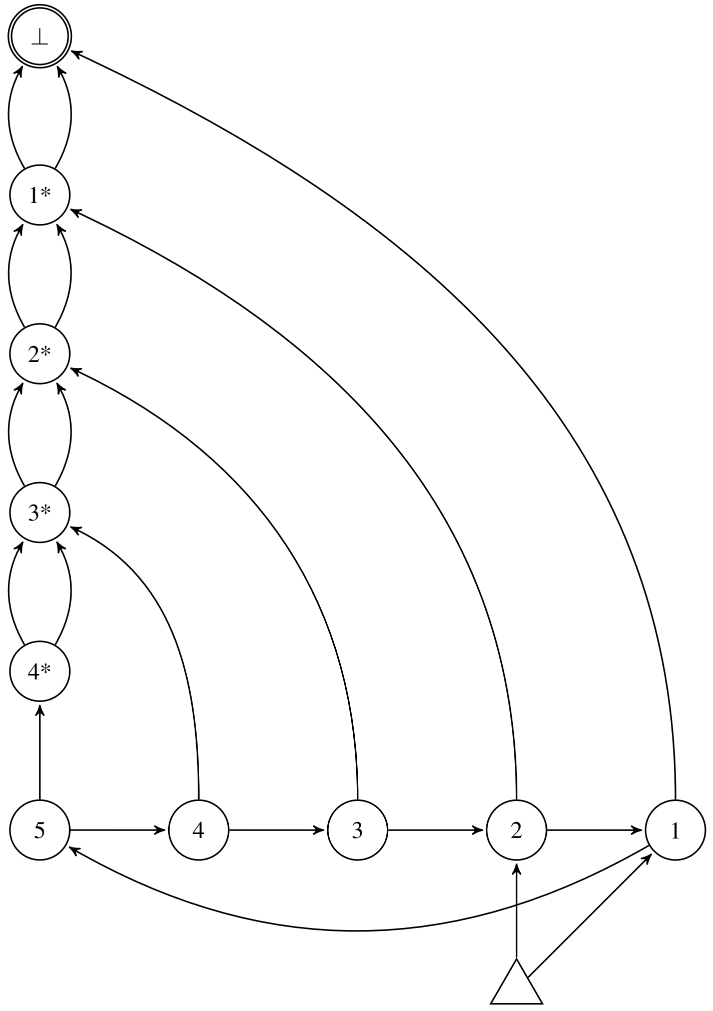


Fig. 3. The MDP G_5. Circle vertices are the coin toss states. The triangle vertex is the Max state. The vertex $\perp$ is the terminal state.

– State i^* has state $(i-1)^*$ as both its successors.
– State i has state $(i-1)^*$ and $(i-1)$ as successors, except state 1 which has $\perp$ and state p as successors.
– The max state has 1 and 2 as successors.

Lemma 4. *Let $p \geq 2$ be given. State i has value $1 - 2^{-f_i(k)}$ in G_p^k for $k > 0$, where $f_i(k)$ is the function $f_i(k) = \max_{k' \leq k \wedge k' \bmod p = i}(k', 0)$.*

Proof. It is easily seen by induction that i^* has value 1 in G_p^i. Note that $f_i(k) = i$ for $k \bmod p = i$. The proof will be by induction in k. There will be one base case and two induction cases, one for $1 < k \leq p$ and one for $k > p$. It is easy to see that state 1 has value $\frac{1}{2} = 1 - \frac{1}{2} = 1 - 2^{-f_1(1)}$ in G_p^1 and state j for $j \neq 1$ has value 0. That settles the base case.

For $1 < k \leq p$. Neither of the successors of state j, for $j \neq k$, has changed values from G_p^{k-2} to G_p^{k-1}. For state k, both its successors has changed value. The value of state $k - 1^*$ has become $\mathrm{val}(G_p^{k-1})_{k-1^*} = 1$ and the value of state $k - 1$ has become $\mathrm{val}(G_p^{k-1})_{k-1} = 1 - 2^{-f_{k-1}(k-1)}$. The value of state k is then

$$\mathrm{val}(G_p^k)_k = \frac{1 + 1 - 2^{-f_{k-1}(k-1)}}{2} = \frac{1 + 1 - 2^{-(k-1)}}{2} = 1 - 2^{-(k-1)-1} = 1 - 2^{-f_k(k)}.$$

For $p < k$. Let i be $k \bmod _p$. Neither of the successors of state j, for $j \neq i$, has changed values from G_p^{k-2} to G_p^{k-1}. The value of state $i' = i - 1 \bmod _p$, in iteration $k - 1$ is $\mathrm{val}(G_p^{k-1})_{i'} = 1 - 2^{-f_{i'}(k-1)}$. The value of state i is then

$$\mathrm{val}(G_p^k)_i = \frac{1 + 1 - 2^{-f_{i'}(k-1)}}{2} = \frac{1 + 1 - 2^{-(k-1)}}{2} = 1 - 2^{-(k-1)-1} = 1 - 2^{-f_i(k)}.$$

The desired result follows. □

The idea behind the construction of F_k is that to find the state of the largest value among 1 and 2, in G_p^T, for $p \geq 2$ and $T \geq 1$, we need to know if $T \bmod p = 1$ or not. Let p_i be the i'th smallest prime number. The FMDP F_k is as follows: F_k consists of a copy of G_{p_i} for $i \in \{1, \ldots, k\}$. Let the max state in that copy of G_{p_i} be m_i. There is a illustration of F_2 in Figure 4.

We will now show that all optimal strategies for F_k are subsets of counter-based strategies with a period defined by k. Afterwards we will show that the number of states in F_k can also be expressed in terms of k. At the end we will use those two lemmas to get to our result.

Lemma 5. *Any optimal strategy $\sigma(k, T')$ in F_k is an finite memory counter-based strategies with period $P = \prod_{i \in \{1,\ldots,k\}} p_i$, where p_i is the i'th smallest prime number.*

Proof. Let i be some number in $\{1, \ldots, k\}$. The lone optimal choice for m_i and $T' > 0$ is to use the action that goes to state 1 in G_{p_i} if $T \bmod p_i = 1$ and otherwise to use the action that goes to state 2 in G_{p_i} by Lemma 4. Hence, by the Chinese remainder theorem there are precisely P steps between each time any optimal strategy uses the action that goes to 1 in all m_i's. That is, any optimal strategy must do the same action at least every P steps. Furthermore it is also easy to see that any optimal strategy must do the same at most every P steps, by noting that $T + P \bmod p_i$ is 1 if and only if $T \bmod p_i$ is 1 and again applying Lemma 4. A strategy that does the same every P steps can be expressed by a counter-based strategy with period P, which also uses memory at most P. □

Lemma 6. *The number of states in F_k is $2 \sum_{i \in \{1,\ldots,k\}} p_i$.*

Proof. For any i, G_{p_i} consists of $2p_i$ states. F_k therefore consists of $2 \sum_{i \in \{1,\ldots,k\}} p_i$ states. □

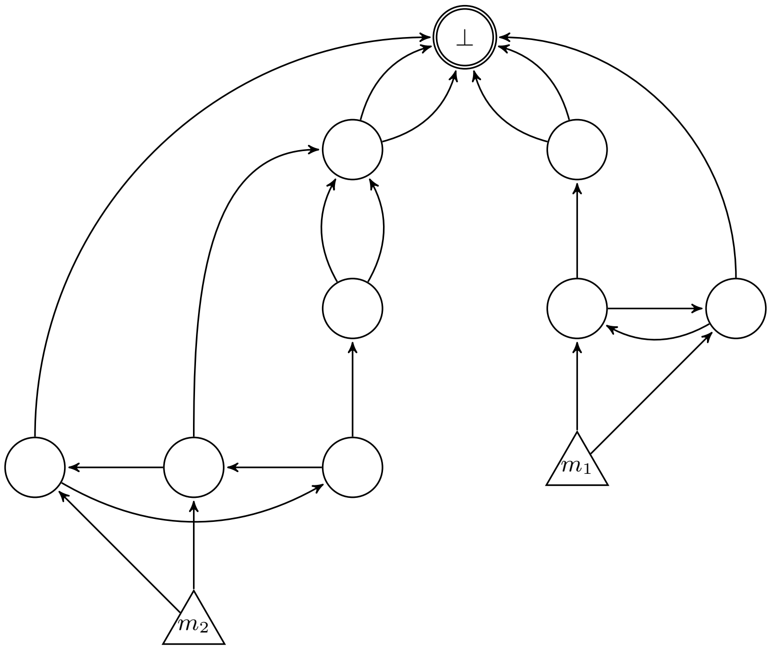


Fig. 4. The FMDP F_2. Circle vertices are the coin toss states. Triangle vertices are the max states. The vertex $\perp$ is the terminal state.

Theorem 3. *There are FMDPs G, with n states, where all optimal strategies are finite memory counter-based strategies with period $2^{\Omega(\sqrt{n \log n})}$.*

Proof. (Sketch). Let n be such that there exists a game F_k with n states. Note that for any number there is always a larger number, a, such that F_b has a states for some b. By Lemma 6, we have that $n = 2\sum_{i\in\{1,...,k\}} p_i$. By the prime number theorem (see e.g. Newman [6]) we have that $\sum_{i\in\{1,...,k\}} p_i = \sum_{i\in\{1,...,k\}} o(k \log k) = o(k^2 \log k)$. Let $f(x) = x^2 \log x$ for $x > 1$. The function $f(x)$ is strictly monotone increasing and hence, has an inverse function. Let that function be $f^{-1}(y)$. We have that $f^{-1}(y) \geq \sqrt{\frac{y}{\log y}}$, for $y \geq 2$. Therefore, let $g(k) = 2\sum_{i\in\{1,...,k\}} p_i$, then $g^{-1}(n) = \Omega(\sqrt{\frac{n}{\log n}})$. By Lemma 5, we have that the period is $\prod_{i\in\{1,...,k\}} p_i$. Trivially we have that $\prod_{i\in\{1,...,k\}} p_i \geq \prod_{i\in\{1,...,k\}} i = k! = 2^{\Omega(k \log k)}$. We now insert $\Omega(\sqrt{\frac{n}{\log n}})$ in place of k and get the desired result. □

5 Conclusion

In the present paper we have considered properties of finite-horizon Markov decision processes and simple stochastic games. The ϵ-optimal strategies considered in Section 3 indicates the hardness of playing such games with a short horizon. The concept of period from Section 4 indicates the hardness of playing such games with a long horizon. Along with our lower bound from Section 4 we conjecture the following:

Conjecture 1. All FSSGs have an optimal strategy, which is an finite memory counter-based strategy, with period at most 2^n.

References

1. Condon, A.: The complexity of stochastic games. Information and Computation 96, 203–224 (1992)
2. Everett, H.: Recursive games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games Vol. III. Annals of Mathematical Studies, vol. 39. Princeton University Press (1957)
3. Filar, J., Vrieze, K.: Competitive Markov Decision Process, ch. 2.2, pp. 16–22. Springer (1997)
4. Howard, R.A.: Dynamic Programming and Markov Processes. M.I.T. Press (1960)
5. Ibsen-Jensen, R., Miltersen, P.B.: Solving simple stochastic games with few coin toss positions. European Symposia on Algorithms (to appear, 2012)
6. Newman, D.J.: Simple analytic proof of the prime number theorem. The American Mathematical Monthly 87(9), 693–696 (1980)
7. Puterman, M.L.: Markov Decision Processes, ch. 4, pp. 74–118. John Wiley & Sons, Inc. (2008)

Controllable-Choice Message Sequence Graphs

Martin Chmelík[1,⋆] and Vojtěch Řehák[2,⋆⋆]

[1] Institute of Science and Technology Austria (IST Austria), Klosterneuburg, Austria
martin.chmelik@ist.ac.at
[2] Faculty of Informatics Masaryk University, Brno, Czech Republic
rehak@fi.muni.cz

Abstract. We focus on the realizability problem of Message Sequence Graphs (MSG), i.e. the problem whether a given MSG specification is correctly distributable among parallel components communicating via messages. This fundamental problem of MSG is known to be undecidable. We introduce a well motivated restricted class of MSG, so called controllable-choice MSG, and show that all its models are realizable and moreover it is decidable whether a given MSG model is a member of this class. In more detail, this class of MSG specifications admits a deadlock-free realization by overloading existing messages with additional bounded control data. We also show that the presented class is the largest known subclass of MSG that allows for deadlock-free realization.

1 Introduction

Message Sequence Chart (MSC) [16] is a popular formalism for specification of distributed systems behaviors (e.g. communication protocols or multi-process systems). Its simplicity and intuitiveness come from the fact that an MSC describes only exchange of messages between system components, while other aspects of the system (e.g. content of the messages and internal computation steps) are abstracted away. The formalism consists of two types of charts: (1) basic Message Sequence Charts (bMSC) that are suitable for designing finite communication patterns and (2) High-level Message Sequence Charts (HMSC) combining bMSC patterns into more complex designs. In this paper, we focus on the further type reformulated as Message Sequence Graphs (MSG) that has the same expressive power as HMSC but a simpler structure, and hence it is often used in theoretical computer science papers, see, e.g. [2,4,6,14,23].

Even such incomplete models as MSG can indicate serious errors in the designed system. The errors can cause problems during implementation or even make it impossible. Concerning verification of MSC models, researchers have studied a presence of a race condition in an MSC [3,7,11,23], boundedness of

⋆ The author was supported by Austrian Science Fund (FWF) Grant No P 23499-N23 on Modern Graph Algorithmic Techniques in Formal Verification, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

⋆⋆ The author is supported by the Czech Science Foundation, grant No. P202/12/G061.

the message channels [4], the possibility to reach a non-local branching node [6,20,14,17,18,12,21], deadlocks, livelocks, and many more. For a recent overview of current results see, e.g. [10].

In this paper, we focus on the realizability problem of MSG specifications, i.e. implementation of the specification among parallel machines communicating via messages. This problem has been studied in various settings reflecting parameters of the parallel machines, the environment providing message exchanges as well as the type of equivalence considered between the MSG specification and its implementation. Some authors restricted the communication to synchronous handshake [14,13], needed several initial states in the synthesized machines [5], or considered language equivalence with global accepting states in the implementation (the implementation accepts if the components are in specific combinations of its states) [22]. From our point of view, the crucial aspect is the attitude to non-accepted executions of the implementation. When language equivalence is taken into account, an intentional deadlock can prevent a badly evolving execution from being accepted [14]. In our setting every partial execution can be extended into an accepting one. Therefore, we focus on a deadlock-free implementation of a given MSG into Communicating Finite-State Machines (CFM) with FIFO communicating channels and distributed acceptance condition, i.e. a CFM accepts if each machine is in an accepting state. In [19], it has been shown that existence of a CFM realizing a given MSG without deadlocks is undecidable. When restricted to *bounded* MSG (aka regular MSG, i.e. communicating via finite/bounded channels, and so generating a regular language), the problem is EXPSPACE-complete [19].

In later work [14,5], a finite data extension of messages was considered when realizing MSG. This is a very natural concept because message labels in MSG are understood as message types abstracting away from the full message content. Hence, during implementation, the message content can be refined with additional (finite) data that helps to control the computation of the CFM in order to achieve the communication sequences as specified in the given MSG. The main obstacle when realizing MSG are nodes with multiple outgoing edges — choice nodes. In a CFM realization, it is necessary to ensure that all Finite-State Machines choose the same successor of each choice node. This can be hard to achieve as the system is distributed.

In [14], a class of so called *local-choice* MSG [18,6] was shown to include only MSG realizable in the above mentioned setting. Local-choice specifications have the communication after each choice node initiated by a single process — the choice leader. Intuitively, whenever a local-choice node is reached, the choice leader machine attaches to all its outgoing messages the information about the chosen node. The other machines pass the information on. This construction is sufficient to obtain a deadlock-free realization, for details see [14]. Another possible realization of local-choice MSG is presented in [17]. Due to [12], it is also decidable to determine whether a given MSG is language equivalent to some local-choice MSG and, moreover, each equivalent MSG can be algorithmically realized by a CFM. To the best of our knowledge, this is the largest class of deadlock-free realizable specifications in the standard setting, i.e. with additional data, FIFO channels, and local accepting states.

In this paper, we introduce a new class of *controllable-choice* MSG that extends this large class of realizable MSG. The crucial idea of controllable-choice MSG is that even some non-local-choice nodes can be implemented, if the processes initiating the communication after the choice can agree on the successor node in advance. This is achieved by exchanging bounded additional content in existing messages. We call choice nodes where such an agreement is possible *controllable-choice* nodes, and show that the class of MSG with these nodes is more expressive than the class of MSG that are language equivalent to local-choice MSG.

2 Preliminaries

In this section, we introduce the Message Sequence Chart (MSC) formalism that was standardized by the International Telecommunications Union (ITU-T) as Recommendation Z.120 [16]. It is used to model interactions among parallel components in a distributed environment. First, we introduce the basic MSC.

Basic Message Sequence Charts (bMSC). Intuitively, a bMSC identifies a single finite execution of a message passing system. Processes are denoted as vertical lines — instances. Message exchange is represented by an arrow from the sending process to the receiving process. Every process identifies a sequence of actions — sends and receives — that are to be executed in the order from the top of the diagram. The communication among the instances is not synchronous and can take arbitrarily long time.

Definition 1. *A* basic Message Sequence Chart (bMSC) *M is defined by a tuple $(E, <, \mathcal{P}, \mathcal{T}, P, \mathcal{M}, l)$ where:*
- *E is a finite set of events,*
- *$<$ is a partial ordering on E called visual order,*
- *$\mathcal{P}$ is a finite set of processes,*
- *$\mathcal{T} : E \to \{send, receive\}$ is a function dividing events into sends and receives,*
- *$P : E \to \mathcal{P}$ is a mapping that associates each event with a process,*
- *$\mathcal{M} : \mathcal{T}^{-1}(\text{send}) \to \mathcal{T}^{-1}(\text{receive})$ is a bijective mapping, relating every send with a unique receive, such that a process cannot send a message to itself, we refer to a pair of events $(e, \mathcal{M}(e))$ as a* message, *and*
- *l is a function associating with every message (e, f) a label m from a finite set of message labels $\mathcal{C}$, i.e. $l(e, f) = m$.*

Visual order $<$ is defined as the reflexive and transitive closure of $\mathcal{M} \cup \bigcup_{p\in\mathcal{P}} <_p$ where $<_p$ is a total order on $P^{-1}(p)$.

We require the bMSC to be *first-in-first-out (FIFO)*, i.e., the visual order satisfies for all messages $(e, f), (e', f')$ and processes p, p' the following condition

$$e <_p e' \wedge P(f) = P(f') = p' \;\Rightarrow\; f <_{p'} f'.$$

Every event of a bMSC can be represented by a letter from an alphabet

$$\Sigma = \{p!q(m) \mid p, q \in \mathcal{P}, m \in \mathcal{C}\} \cup \{q?p(m) \mid p, q \in \mathcal{P}, m \in \mathcal{C}\}.$$

Intuitively, $p!q(m)$ denotes a send event of a message with a label m from a process p to a process q, and $q?p(m)$ represents a receive event of a message with a label m by q from a process p. We define a *linearization* as a word over Σ representing a total order of events that is consistent with the partial order $<$. For a given bMSC M, a *language* $\mathcal{L}(M)$ is the set of all linearizations of M.

Message Sequence Graphs. It turns out that specifying finite communication patterns is not sufficient for modelling complex systems. Message Sequence Graphs allow us to combine bMSCs into more complex systems using alternation and iteration. An MSG is a directed graph with nodes labeled by bMSCs and two special nodes, the initial and the terminal node. Applying the concept of finite automata [15], the graph represents a set of paths from the initial node to the terminal node. In MSG, every such a path identifies a sequence of bMSCs. As every finite sequence of bMSCs can be composed into a single bMSC, an MSG serves as a finite representation of an (infinite) set of bMSCs.

Definition 2. *A* Message Sequence Graph (MSG) *is defined by a tuple* $G = (\mathcal{S}, \tau, s_0, s_f, L)$, *where* $\mathcal{S}$ *is a finite set of states,* $\tau \subseteq \mathcal{S} \times \mathcal{S}$ *is an edge relation,* $s_0 \in \mathcal{S}$ *is the initial state,* $s_f \in \mathcal{S}$ *is the terminal state, and* $L : \mathcal{S} \to$ bMSC *is a labeling function.*

W.l.o.g., we assume that there is no incoming edge to s_0 and no outgoing edge from s_f. Moreover, we assume that there are no nodes unreachable from the initial node and the terminal node is reachable from every node in the graph.

Given an MSG $G = (\mathcal{S}, \tau, s_0, s_f, L)$, a *path* is a finite sequence of states $s_1 s_2 \dots s_k$, where $\forall\, 1 \le i < k : (s_i, s_{i+1}) \in \tau$. A *run* is defined as a path with $s_1 = s_0$ and $s_k = s_f$.

Intuitively, two bMSCs can be composed to a single bMSC by appending events of every process from the latter bMSC at the end of the process from the precedent bMSC. Formally, the *sequential composition* of two bMSCs $M_1 = (E_1, <_1, \mathcal{P}, \mathcal{T}_1, P_1, \mathcal{M}_1, l_1)$ and $M_2 = (E_2, <_2, \mathcal{P}, \mathcal{T}_2, P_2, \mathcal{M}_2, l_2)$ such that the sets E_1 and E_2 are disjoint (we can always rename events so that the sets become disjoint), is the bMSC $M_1 \cdot M_2 = (E_1 \cup E_2, <, \mathcal{P}, \mathcal{T}_1 \cup \mathcal{T}_2, P_1 \cup P_2, \mathcal{M}_1 \cup \mathcal{M}_2, l_1 \cup l_2)$, where $<$ is a transitive closure of $<_1 \cup <_2 \cup \bigcup_{p \in \mathcal{P}} (P_1^{-1}(p) \times P_2^{-1}(p))$. Note that we consider the weak concatenation, i.e. the events from the latter bMSC may be executed even before some events from the precedent bMSC.

Now, we extend the MSG labeling function L to paths. Let $\sigma = s_1 s_2 \dots s_n$ be a path in MSG G, then $L(\sigma) = L(s_1) \cdot L(s_2) \cdot \ldots \cdot L(s_n)$. For a given MSG G, the *language* $\mathcal{L}(G)$ is defined as $\bigcup_{\sigma \text{ is a run in } G} \mathcal{L}(L(\sigma))$. Hence, two MSG are said to be *language-equivalent* if and only if they have the same languages.

Communicating Finite-State Machines. A natural formalism for implementing bMSCs are Communicating Finite–State Machines (CFM) that are used for example in [3,1,14]. The CFM consists of a finite number of finite-state machines that communicate with each other by passing messages via unbounded FIFO channels.

Definition 3. *Given a finite set $\mathcal{P}$ of processes and a finite set of message labels $\mathcal{C}$, the Communicating Finite-State Machine (CFM) $\mathcal{A}$ consists of Finite-State Machines (FSMs) $(\mathcal{A}_p)_{p\in\mathcal{P}}$. Every $\mathcal{A}_p$ is a tuple $(\mathcal{S}_p, A_p, \rightarrow_p, s_p, F_p)$, where:*
- *$\mathcal{S}_p$ is a finite set of states,*
- *$A_p \subseteq \{p!q(m) \mid q \in \mathcal{P}, m \in \mathcal{C}\} \cup \{p?q(m) \mid q \in \mathcal{P}, m \in \mathcal{C}\}$ is a set of actions,*
- *$\rightarrow_p \subseteq \mathcal{S}_p \times A_p \times \mathcal{S}_p$ is a transition relation,*
- *$s_p \in \mathcal{S}_p$ is the initial state, and*
- *$F_p \subseteq \mathcal{S}_p$ is a set of local accepting states.*

We associate an unbounded FIFO error-free channel $\mathcal{B}_{p,q}$ with each pair of FSMs $\mathcal{A}_p, \mathcal{A}_q$. In every configuration, the content of the channel is a finite word over the label alphabet $\mathcal{C}$.

Whenever an FSM $\mathcal{A}_p$ wants to send a message with a label $m \in \mathcal{C}$ to $\mathcal{A}_q$, it enqueues the label m into channel $\mathcal{B}_{p,q}$. We denote this action by $p!q(m)$. Provided there is a message with a label m in the head of channel $\mathcal{B}_{p,q}$, the FSM $\mathcal{A}_q$ can receive and dequeue the message with the label m. This action is represented by $q?p(m)$. A *configuration* of a CFM $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathcal{P}}$ is a tuple $C = (s, B)$, where $s \in \prod_{p\in\mathcal{P}}(\mathcal{S}_p)$ and $B \in (\mathcal{C}^*)^{\mathcal{P}\times\mathcal{P}}$ — local states of the FSMs together with the contents of the channels. Whenever there is a configuration transition $C_i \xrightarrow{a_i} C_{i+1}$, there exists a process $p \in \mathcal{P}$ such that the FSM $\mathcal{A}_p$ changes its local state by executing action $a_i \in A_p$ and modifies the content of one of the channels.

The CFM execution starts in an *initial configuration* $s_0 = \prod_{p\in\mathcal{P}}\{s_p\}$ with all the channels empty. The CFM is in an *accepting configuration*, if every FSM is in some of its final states and all the channels are empty. We will say that a configuration is a *deadlock*, if no accepting configuration is reachable from it. A CFM is *deadlock-free* if no deadlock configuration is reachable from the initial configuration. An *accepting execution* of a CFM $\mathcal{A}$ is a finite sequence of configurations $C_1 \xrightarrow{a_1} C_2 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} C_n$ such that C_1 is the initial configuration and C_n is an accepting configuration. The word $a_1a_2\cdots a_{n-1}$ is then an *accepted word* of $\mathcal{A}$. Given a CFM $\mathcal{A}$, the *language* $\mathcal{L}(\mathcal{A})$ is defined as the set of all accepted words of $\mathcal{A}$.

3 Controllable-Choice Message Sequence Graphs

For a given MSG we try to construct a CFM such that every execution specified in the MSG specification can be executed by the CFM and the CFM does not introduce any additional unspecified execution.

Definition 4 ([1]). *An MSG G is* realizable *iff there exists a deadlock-free CFM $\mathcal{A}$ such that $\mathcal{L}(G) = \mathcal{L}(\mathcal{A})$.*

One of the most natural realizations are projections. A projection of a bMSC M on a process p, denoted by $M|_p$, is the sequence of events that are to be executed by the process p in M. For every process $p \in \mathcal{P}$, we construct a FSM $\mathcal{A}_p$ that accepts a single word $M|_p$. This construction is surprisingly powerful and models all of the bMSC linearizations.

Proposition 1. *Let M be a bMSC, then CFM $\mathcal{A} = (M|_p)_{p \in \mathcal{P}}$ is a realization, i.e. $\mathcal{L}(M) = \mathcal{L}(\mathcal{A})$.*

It turns out that the main obstacle when realizing MSG are nodes with multiple outgoing edges — choice nodes. It is necessary to ensure that all FSMs choose the same run through the MSG graph. This can be hard to achieve as the system is distributed.

In what follows, we present a known class of local-choice MSG specifications that admits a deadlock-free realization by adding control data into the messages. Then, we define a new class of *controllable-choice* MSG and compare the expressive power of the presented classes.

Local-Choice MSG. is a Class Studied by Many Authors [6,20,14,17,18,12]. Let M be a bMSC, we say that a process $p \in \mathcal{P}$ *initiates* the bMSC M if there exists an event e in M, such that $P(e) = p$ and there is no other event e' in bMSC M such that $e' < e$. For a given MSG, every node $s \in \mathcal{S}$ identifies a set *triggers*(s), the set of processes initiating the communication after the node s. Note that it may not be sufficient to check only the direct successor nodes in the MSG.

Definition 5. *Let $G = (\mathcal{S}, \tau, s_0, s_f, L)$ be an MSG. For a node $s \in \mathcal{S}$, the set* triggers(s) *contains process p if and only if there exists a path $\sigma = \sigma_1 \sigma_2 \ldots \sigma_n$ in G such that $(s, \sigma_1) \in \tau$ and p initiates bMSC $L(\sigma)$.*

Definition 6. *A choice node u is a* local-choice *node iff* triggers(u) *is a singleton. An MSG specification G is* local-choice *iff every choice node of G is local-choice.*

Local-choice MSG specifications have the communication after every choice node initiated by a single process — the choice leader. In [14] a deadlock-free realization with additional data in messages is proposed. It is easy to see that every MSG specification G is deadlock-free realizable if there is a local-choice MSG G' such that $\mathcal{L}(G) = \mathcal{L}(G')$. Note that the equivalence can be algorithmically checked due to [12].

Controllable Specifications. The difficulties when realizing MSG are introduced by choice nodes. In local-choice MSG, the additional message content is used to ensure a single run through the graph is executed by all FSMs. In case of controllable-choice MSG, the additional content serves the same purpose but besides informing about the node the FSMs are currently executing the FSMs also attach a prediction about its future execution.

This allows us to relax the restriction on choice nodes and allows certain non-local choice nodes to be present in the specification. However, it is necessary to be able to resolve every occurrence of the choice node, i.e. make the decision in advance and inform all relevant processes.

Definition 7. *Let $M = (E, <, \mathcal{P}, \mathcal{T}, P, \mathcal{M}, l)$ be a bMSC and $P' \subseteq \mathcal{P}$ be a subset of processes. A send event $e \in E$ is a* resolving event *for P' iff*

$$\forall p \in P' \; \exists e_p \in P^{-1}(p) \text{ such that } e < e_p.$$

Intuitively, resolving events of M for P' can distribute information to all processes of P' while executing the rest of M, provided that other processes are forwarding the information.

Definition 8. *Let $G = (\mathcal{S}, \tau, s_0, s_f, L)$ be an MSG. A choice node u is said to be* controllable-choice *iff it satisfies both of the following conditions:*

- *For every path σ from s_0 to u there exists a resolving event in bMSC $L(\sigma)$ for* triggers(u).
- *For every path $\sigma = s_1 s_2 \dots u$ such that $(u, s_1) \in \tau$, there exists a resolving event in bMSC $L(\sigma)$ for* triggers(u)*.*

Intuitively, a choice node is controllable-choice, if every path from the initial node is labeled by a bMSC with a resolving event for all events initiating the communication after branching. Moreover, as it is necessary to attach only bounded information, the same restriction is required to hold for all cycles containing a controllable-choice node. In [9] we propose an algorithm that determines whether a given choice node is a controllable-choice node.

Definition 9. *An MSG specification G is* controllable-choice *iff every choice node is either local-choice or controllable.*

Note that there is no bound on the distance between the resolving event and the choice node it is resolving.

Local-Choice vs. Controllable-Choice MSG. In the following, we show that the controllable-choice MSG are more expressive than local-choice MSG. It is easy to see that every local-choice MSG is also a controllable-choice MSG and that not every controllable-choice MSG is local-choice. In the following theorem, we strengthen the result by stating that the class of MSG that are language equivalent to some controllable-choice MSG is more expressive than the class of MSG that are language-equivalent to some local-choice MSG.

Theorem 1. *The class of MSG that are language-equivalent to some local-choice MSG, forms a proper subset of MSG that are language-equivalent to some controllable-choice MSG.*

Proof. Consider a MSG $G = (\mathcal{S}, \tau, s_0, s_f, L)$ with three nodes s_0, s_f and s, such that $(s_0, s), (s, s), (s, s_f) \in \tau$ and the only non–empty bMSC is $L(s)$ with two processes p, q. The projection of events on p is $p!q(m), p?q(m')$ and similarly for q the projection is $q!p(m'), q?p(m)$. Note that the only choice node s is controllable as both send events are resolving events for both of the processes.

The MSG G violates a necessary condition to be language equivalent to a local-choice specification. Intuitively, the condition states that its language must be a subset of a language of a generic local-choice equivalent MSG (for more details see [12]).

4 Realizability of Controllable-Choice MSG

In this section we present an algorithm for realization of controllable-choice MSG. The class of local-choice specifications admits a natural deadlock-free realization because every branching is controlled by a single process.

As the *triggers* set for controllable-choice nodes can contain multiple processes, we need to ensure that all of them reach a consensus about which branch to choose. To achieve this goal, we allow the FSMs in certain situations to add a behavior prediction into its outgoing messages. Those predictions are stored in the finite-state control units and are forwarded within the existing communication to other FSMs.

The length of the prediction should be bounded, as we can attach only bounded information to the messages and we need to store it in the finite-state control unit. Therefore, it may be necessary to generate the behavior predictions multiple times. As the realization should be deadlock-free, we must ensure that the predictions are not conflicting — generated concurrently by different FSMs. To solve this we sometimes send together with the prediction also an event where the next prediction should be generated.

Definition 10. *A* prediction *for an MSG $G = (\mathcal{S}, \tau, s_0, s_f, L)$ is a pair $(\sigma, e) \in \mathcal{S}^* \times (E \cup \bot)$, where E is the set of all events of bMSCs assigned by L, the path σ is called a* prediction path*, and e, called a* control event*, is an event from $L(\sigma)$. A prediction path must satisfy one of the following conditions:*

- *The prediction path σ is the longest common prefix of all MSG runs. This special initial prediction path is named $initialPath$.*
- *The prediction path σ is the shortest path $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ in G satisfying*
  1. *$\sigma_n \in \mathcal{L}$, or*
  2. *$\sigma_n \in \mathcal{U} \wedge \exists\, 1 \leq i < n : \sigma_i = \sigma_n$, or*
  3. *$\sigma_n = s_f$,*

  where $\mathcal{L} \subseteq \mathcal{S}$ is the set of all local-choice nodes and $\mathcal{U} \subseteq \mathcal{S}$ is the set of all controllable-choice nodes.

We refer to the first node and to the last node of a prediction path σ by *firstNode*(σ) and *lastNode*(σ), respectively.

Lemma 1. *If the prediction path σ ends with a controllable-choice node u, the bMSC $L(\sigma)$ contains a resolving event for* triggers(u) *on $L(\sigma)$.*

Proof. There are two cases to consider

- If $\sigma = \mathit{initialPath}$, then $\mathit{firstNode}(\sigma) = s_0$ and as node u is controllable-choice, the path σ contains a resolving event for $\mathit{triggers}(u)$.
- Otherwise, the controllable-choice node u occurs twice in the path σ. As every cycle containing a controllable-choice node has to contain a resolving event for the node, there is a resolving event for $\mathit{triggers}(u)$ on path σ.

As there are no outgoing edges allowed in s_f, the terminal node $s_f \notin \mathcal{U}$. $\square$

Note, that the number of events in a given MSG is finite and the length of each prediction path is bounded by $2 \cdot |\mathcal{S}|$.

When the CFM execution starts, every FSM is initialized with an initial prediction — $(\mathit{initialPath}, e_i)$ — and starts to execute the appropriate projection of $L(\mathit{initialPath})$. The value of e_i depends on the *initialPath*. Let $\mathit{lastNode}(\mathit{initialPath}) = \sigma_n$. In case of $\sigma_n \in \mathcal{U}$, the event e_i is an arbitrary resolving event from $L(\mathit{initialPath})$ for $\mathit{triggers}(\sigma_n)$. It follows from Lemma 1 that there exists such an event. If $\sigma_n \in \mathcal{L} \cup \{s_f\}$, we set $e_i = \perp$.

Every FSM stores two predictions, one that is being currently executed and a future prediction that is to be executed after the current one. Depending on the *lastNode* of the current prediction, there are the following possibilities where to generate the future prediction.

- If *lastNode* of the current prediction is in $\mathcal{L}$, the future prediction is generated by the local-choice leader, while executing the first event after branching.
- If *lastNode* of the current prediction is in $\mathcal{U}$, the future prediction is generated by an FSM that executes the control event of the current prediction, while executing the resolving event.
- If the *lastNode* of the current prediction is s_f, no further execution is possible and so no new prediction is generated.

When an FSM generates a new prediction, we require that there exists a transition in the MSG from the last node of the current prediction path to the first node of the future prediction path, as the concatenation of prediction paths should result in a path in the MSG. If an FSM generates a future prediction ending with a controllable-choice node u, it chooses an arbitrary resolving event for $\mathit{triggers}(u)$ to be the resolving event in the prediction. The existence of such an event follows from Lemma 1. To ensure that other FSMs are informed about the decisions, both predictions are attached to every outgoing message. The computation ends when no FSM is allowed to generate any future behavior.

4.1 Algorithm

In this section, we describe the realization algorithm. All the FSMs execute the same algorithm, an implementation of the FSM $\mathcal{A}_p$ is described in Algorithm 1. We use an auxiliary function **path** that returns a prediction path for a given prediction. Every FSM stores a queue of events that it should execute — *eventQueue*. The queue is filled with projections of bMSCs labeling projection paths — $L(\text{prediction path})|_p$ for FSM $\mathcal{A}_p$. The execution starts with filling the queue with the projection of the *initialPath*.

Algorithm 1. Process p implementation

```
1: Variables: $currentPrediction, nextPrediction, eventQueue$;
2: $currentPrediction \leftarrow (initialPath, e_i)$;
3: $nextPrediction \leftarrow \perp$;
4: $eventQueue \leftarrow$push$(L(initialPath)|_p)$;
5: while true do
6:    if $eventQueue$ is empty then
7:       getNextNode();
8:    $e \leftarrow$ pop$(eventQueue)$;
9:    if $e$ is a send event then
10:       if $e$ is the resolving event in $currentPrediction$ then
11:          $node \leftarrow$ lastNode(path$(currentPrediction)$);
12:          $nextPrediction \leftarrow$ guessPrediction$(node)$;
13:       send$(e, currentPrediction, nextPrediction)$;
14:    if $e$ is a receive event then
15:       receive$(e, cP, nP)$;
16:       if $nextPrediction = \perp$ then
17:          $nextPrediction \leftarrow nP$;
```

Function 2. getNextNode function for process p

```
1: Function getNextNode()
2:    $node \leftarrow$ lastNode(path$(currentPrediction)$);
3:    if $node \in \mathcal{U} \wedge p \in$ triggers$(node)$ then
4:       $currentPrediction \leftarrow nextPrediction$;
5:       $nextPrediction \leftarrow \perp$;
6:       $eventQueue \leftarrow$push$(L($path$(currentPrediction))|_p)$;
7:    else if $node \in \mathcal{L} \wedge p \in$ triggers$(node)$ then
8:       $currentPrediction \leftarrow$ guessPrediction$(node)$;
9:       $nextPrediction \leftarrow \perp$
10:      $eventQueue \leftarrow$push$(L($path$(currentPrediction))|_p)$;
11:   else
12:      $currentPrediction \leftarrow \perp$;
13:      $nextPrediction \leftarrow \perp$;
14:      polling();
15: end function
```

Function 3. Polling function for process p

```
1: Function polling()
2:    while true do
3:       if $p$ has a message in some of its input buffers then
4:          receive$(e, cP, nP)$;
5:          $currentPrediction \leftarrow cP$;
6:          $nextPrediction \leftarrow nP$;
7:          $eventQueue \leftarrow$push$(L($path$(currentPrediction))|_p)$;
8:          pop$(eventQueue)$;
9:          return;
10: end function
```

The FSM executes a sequence of events according to its *eventQueue*. In order to exchange information with other FSMs, it adds its knowledge of predictions to every outgoing message, and improves its own predictions by receiving messages from other FSMs.

When the FSM executes a control event of the current prediction, it is responsible for generating the next prediction. The function **guessPrediction**(u) behaves as described in the previous section. It chooses a prediction (σ, e), such that $(u, \text{firstNode}(\sigma)) \in \tau$. If $\mathit{lastNode}(\sigma) \in \mathcal{U}$, then e is a chosen resolving event in bMSC $L(\sigma)$ for the *triggers* set of the $\mathit{lastNode}(\sigma)$. Otherwise, we leave $e = \bot$.

If the *eventQueue* is empty, the FSM runs the **getNextNode** function to determine the continuation of the execution. If the *lastNode* of the current prediction is a controllable-choice node and p is in the *triggers* set of this node, it uses the prediction from its variable *nextPrediction* as its *currentPrediction*. The variable *nextPrediction* is set to $\bot$.

If the *lastNode* of the currentPrediction is a local-choice node and p is the leader of the choice, it guesses the prediction and assigns it to the appropriate variables. Otherwise, the FSM forgets its predictions and enters a special polling state. This state is represented by the **Polling** function. Whenever the FSM receives a message, it sets its predictions according to the message. The pop function on line 8 ensures the consistency of the *eventQueue*.

An execution is finished successfully if all the FSMs are in the polling state and all the buffers are empty. The correctness proof of the following theorem can be found in the full version [8].

Theorem 2. *Let G be a controllable-choice MSG. Then the CFM $\mathcal{A}$ constructed by Algorithm 1 is a deadlock-free realization i.e. $\mathcal{L}(G) = \mathcal{L}(\mathcal{A})$.*

5 Conclusion

In this work we studied the message sequence graph realizability problem, i.e., the possibility to make an efficient and correct distributed implementation of the specified system. In general, the problem of determining whether a given specification is realizable is undecidable. Therefore, restricted classes of realizable specifications are in a great interest of software designers.

In recent years, a promising research direction is to study deadlock-free realizability allowing to attach bounded control data into existing messages. This concept turns out to be possible to realize reasonable specifications that are not realizable in the very original setting. In this work we introduced a new class of so called controllable-choice message sequence graphs that admits a deadlock-free realization with additional control data in messages. In other words, we have sucesfully extended the class of MSG conforming in the established setting of realizability. Moreover, we have presented an algorithm producing realization for a given controllable-choice message sequence graphs.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. In: Proc. of ICSE, pp. 304–313 (2000)
2. Alur, R., Etessami, K., Yannakakis, M.: Realizability and verification of MSC graphs. Theor. Comp. Science 331(1), 97–114 (2005)
3. Alur, R., Holzmann, G.J., Peled, D.: An Analyzer for Message Sequence Charts. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 35–48. Springer, Heidelberg (1996)
4. Alur, R., Yannakakis, M.: Model Checking of Message Sequence Charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)
5. Baudru, N., Morin, R.: Safe implementability of regular message sequence chart specifications. In: Proc. of ACIS, pp. 210–217 (2003)
6. Ben-Abdallah, H., Leue, S.: Syntactic Detection of Process Divergence and Non-local Choice in Message Sequence Charts. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 259–274. Springer, Heidelberg (1997)
7. Chen, C.-A., Kalvala, S., Sinclair, J.E.: Race Conditions in Message Sequence Charts. In: Yi, K. (ed.) APLAS 2005. LNCS, vol. 3780, pp. 195–211. Springer, Heidelberg (2005)
8. Chmelík, M., Řehák, V.: Controllable-choice message sequence graphs. CoRR, abs/1209.4499 (2012)
9. Chmelík, M.: Deciding Non–local Choice in High–level Message Sequence Charts Bachelor thesis, Faculty of Informatics, Masaryk University, Brno (2009)
10. Dan, H., Hierons, R.M., Counsell, S.: A framework for pathologies of message sequence charts. Information and Software Technology (in press, 2012)
11. Elkind, E., Genest, B., Peled, D.A.: Detecting Races in Ensembles of Message Sequence Charts. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 420–434. Springer, Heidelberg (2007)
12. Genest, B., Muscholl, A.: The Structure of Local-Choice in High-Level Message Sequence Charts (HMSC). Technical report, LIAFA, Université Paris VII (2002)
13. Genest, B., Muscholl, A., Kuske, D.: A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 30–48. Springer, Heidelberg (2004)
14. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-State High-Level MSCs: Model-Checking and Realizability. Journal of Computer and System Sciences 72(4), 617–647 (2006)
15. Gill, A.: Introduction to the Theory of Finite-state Machines. McGraw-Hill (1962)
16. ITU Telecommunication Standardization Sector Study group 17. ITU recommandation Z.120, Message Sequence Charts, MSC (2004)
17. Jard, C., Abdallah, R., Hélouët, L.: Realistic Implementation of Message Sequence Charts. Rapport de recherche RR-7597, INRIA (April 2011)
18. Ladkin, P.B., Leue, S.: Interpreting Message Flow Graphs. Formal Aspects of Computing 7(5), 473–509 (1995)
19. Lohrey, M.: Realizability of high-level message sequence charts: closing the gaps. Theoretical Computer Science 309(1-3), 529–554 (2003)

20. Mooij, A.J., Goga, N., Romijn, J.M.T.: Non-local Choice and Beyond: Intricacies of MSC Choice Nodes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 273–288. Springer, Heidelberg (2005)
21. Mousavi, A., Far, B., Eberlein, A.: The Problematic Property of Choice Nodes in high-level Message Sequence Charts. Tech. report, University of Calgary (2006)
22. Mukund, M., Narayan Kumar, K., Sohoni, M.: Synthesizing Distributed Finite-State Systems from MSCs. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 521–535. Springer, Heidelberg (2000)
23. Řehák, V., Slovák, P., Strejček, J., Hélouët, L.: Decidable Race Condition and Open Coregions in HMSC. Elect. Communications of the EASST 29, 1–12 (2010)

A Better Way towards Key Establishment and Authentication in Wireless Sensor Networks

Filip Jurnečka and Vashek Matyáš

Faculty of Informatics
Masaryk University
Brno, Czech Republic
{xjurn,matyas}@fi.muni.cz

Abstract. In this paper, we show that a previously published paper proposing both key establishment and node authentication protocols actually fails to provide the much needed security. In particular, we show a number of ways to compromise these protocols. To overcome flaws of these protocols, we propose novel protocols that remedy all the found security problems of the previous ones. Additionally, our proposals are by the state of the art computationally, energy and memory efficient.

1 Introduction

Wireless sensor networks (WSNs) are multi-hop networks composed of low-end devices called nodes or motes usually equipped with sensors monitoring some physical phenomena. Additionally, there can be one or more higher-class devices called base stations (BSs) that serve as managers of the WSN and gateways to other networks. Sensor nodes are generally very limited devices with restricted computational, energy and storage resources. The nodes are often deployed in an unaccessible environment, only powered by a set of batteries and expected to last up to several years. It is known that the most significantly power consuming component of a WSN node is the radio [12].

A critical security issue within WSNs is the issue of key management. A key management scheme [6,5] was presented claiming to have near perfect properties. It uses only a single shared global key during an initialization phase, performs minimum computations and communication by broadcasting a single random nonce for a short period of time and using only symmetric primitives. It results in neighboring nodes sharing pairwise keys and enables safe authentication and key exchange with a newcoming node into the network.

However, we have identified a number of weaknesses that render the protocol very insecure as well as energy inefficient even when considering an attacker weaker then assumed by authors.

The contribution of our paper is twofold:

1. We perform a security and efficiency analysis of a recent scheme proposed in [6] and suggest multiple attack vectors.

2. We propose our own schemes providing the same security objectives and perform the security and efficiency analysis of the proposals.

The paper is organized as follows. We first describe our goals, assumptions under which these goals are to be achieved, and the attacker model we are using in Sect. 2. In Sect. 3 we describe the scheme proposed in [6] and perform a brief analysis and pinpoint its drawbacks in Sect. 4. In Sect. 5, we propose two protocols for key management and node authentication that overcome the weaknesses found in [5] and [6] while supporting delayed joining the network. The efficiency and security of our proposals is discussed in Sect. 6. We conclude the paper with Sect. 7.

1.1 Related Work

Over the last decade various schemes have been suggested. As discussed in [15], they can be classified in three categories, based on the encryption techniques used: symmetric, asymmetric and hybrid. Additionally, unencrypted techniques should be also considered, as Anderson et al. [1] made a stand for the key exchange in clear in their key infection mechanism. Due to the computational, storage and communicational effectiveness, we focus our attention on the symmetric schemes.

In 2001, Perrig et al. [13] proposed the SPINS scheme, where the base station shares a pairwise key with each node in the network. When two nodes need a pairwise key, they request one from the base station. To address the scalability issues, Chan and Perrig [3] proposed the PIKE scheme that distributes the load to third node for each pair of nodes. A master key based protocol called BROSK was proposed in 2002 [10] that was further elaborated on in [9].

Another work by Eschenauer and Gligor [8] (EG) proposed the random key predistribution, where each node is preloaded with a subset of random keys from a large pool. After deployment, nodes try to find a shared key with their neighbours from their subsets of the pool. This work was extended in [4] by Chan et al. by requiring $q > 1$ common keys to be shared instead of just 1. In turn, this approach increases node capture resiliency of the scheme.

Du et al. [7] proposed a pairwise key pre-distribution scheme based on the Blom's matrix-based key scheme [2] and random key pre-distribution scheme. As WSNs can be seen as random graphs, this scheme assigns keys only to connected graph links unlike Blom's complete graph links. Hence this scheme is scalable and more resilient to node capture. Another category is the tree-based predistribution. In 2005 Lee and Stinson [11] presented deterministic multiple space Blom's schemes. By weakening the connectivity of the graph, they improved the resiliency and improved the results of [7]. Recently, [14] presented a hierarchical tree-based key management scheme that supports real-time rekeying to provide resiliency to node capture attack.

2 Goals and Assumptions

In this section we describe the initial goals of our scheme, the attacker model and assumptions taken into consideration while designing the scheme.

In the area of WSN security, there is an abundant list of possible security objectives. These include node capture resiliency, protection against eavesdropping or message modification and others. Preventing unauthorized nodes from monitoring and influencing the communication can be done later by, e.g., encryption and obfuscation. The *primary goals* of our scheme follow:

- Scalability not tightly (linearly) bound to the size of the network.
- Provide node capture resiliency, that is only links with the captured node are compromised or random additional links are compromised.
- Provide neighbouring node to node authentication.
- Allow new nodes to join the network after the deployment phase is over.
- Be communicationally efficient by keeping the length and number of messages to a minimum as well as avoid using multi-hop communication.

As we were inspired by [6], we keep their *assumptions* with the addition of partially secure initialization phase. The list follows:

- No prior knowledge of network topology.
- The initial size and density of the network known in advance but arbitrary.
- Static topology with possible future addition and removal of nodes.
- Usage of symmetric and other computationally efficient cryptography only.
- A single preinstalled network-wide keying material (key) for the initialization phase.
- Partially secure initialization phase of at least several seconds right after the deployment, when the attacker can not acquire the network-wide key.

The assumption of a static network comes from [5] and is actually quite important, yet it was mentioned only in the second publication of the paper. The assumption of partially secure initialization phase also applies to the newly added nodes during lifetime of the network.

2.1 Attacker Model

We assume the "real world" attacker model as defined in [1] with extended property of low attacker presence during network initialization. We apply this property also to the newcoming nodes to the network.

- Attacker is present in the deployment area prior to deployment, but is able to monitor only small areas of the network during initialization phase.
- Attacker can perform all passive and active attacks during the initialization phase except the node capture.
- After the initialization phase, an attacker can become global and execute any attack, including the node capture.

Unlike in [1], we assume the attacker to be able to execute small scale active attacks during the initialization phase, i.e. attacks limited to the areas of the network where the attacker has been already present.

3 Delgado-Mohatar et al. Scheme

The Delgado-Mohatar et al. proposal [6] is a candidate scheme for satisfying our goals. The actual scheme consists of two protocols, the key establishment protocol and the node authentication protocol. In this section, we briefly describe these protocols as proposed by authors.

We use the notation as follows:

Symbol	Meaning	Symbol	Meaning
ID_A	identifier of node A	$h(m)$	hash of message m
n_A	a nonce generated by node A	$h^i(m)$	i chained hashes of message m
$n_A \| n_B$	concatenation of n_A and n_B	$h_k(m)$	MAC of message m under key k
k_M	network-wide master key	$e_k(m)$	encryption of message m under key k
k_A	an encryption key of node A	k^i_{auth}	authentication key in i-th cycle
k_{AB}	a pairwise key betw. A and B	∇^i_A	auth. operator of node A in i-th cycle

3.1 Key Establishment Protocol

The key establishment protocol takes place before and right after deployment of the network. Before the network is distributed to the target location, each node is preloaded with a common network-wide master key k_M.

During the network initialization phase, node A generates a random encryption key k_A. The encryption key is calculated as $k_A = h(k_M|n_A)$. Then, each node broadcasts its random value n_A for a short period of time. All neighboring nodes able to hear this value can compute the encryption key of the sender and use it in future communication.

3.2 Authentication Protocol

This protocol serves for authenticating new nodes joining the network. Delgado-Mohatar et al. propose an authentication protocol based on master key, yet without actually storing it. It is based on so called authentication operators (authenticators for short) and hashes of the master key.

In this paper, we solve ambiguities of the original paper, hence our description is not completely identical to the proposal.

Definition 1. *The authentication key is updated in "cycles" and the i-th cycle is obtained as a hashed master key as $k^i_{auth} = h^i(k_M)$.*

In this manner, the master key can be seen as $k_M = k^0_{auth}$.

Definition 2. *An authenticator ∇^j is a set of x pairs of a random value and an image of it generated by hash protected with an authentication key in j-th cycle:*

$$\nabla^j = \{(n_i, h_{k^j_{auth}}(n_i))\}, i = 0, \dots, x-1.$$

The number of pairs is usually small, e.g., 5.

A node generates ∇^0 and the first authentication key k^1_{auth} and erases the master key k_M. Note that the stored authentication key should always be by one cycle further from the authenticator. That concludes the network initialization phase.

When a new node A wants to join the network, it is loaded with the master key k_M and a classical challenge/response mechanism is employed to authenticate A and to provide it with keys shared with neighbours.

The protocol can be summarized as follows:

- The fresh node A challenges node B with a new random nonce n_A.
- Node B responds with nonce n_B taken as the first unused nonce from ∇^j, image of the challenge $h_{k^{j+1}_{auth}}(n_A)$, its encryption key encrypted under authentication key $e_{k^{j+1}_{auth}}(k_B)$ and the current cycle j.
- Node A computes k^{j+1}_{auth} and verifies the image of n_A, replies with the image of n_B, i.e., $h_{k^j_{auth}}(n_B)$ and its encryption key k_A encrypted under k^{j+1}_{auth} as $e_{k^{j+1}_{auth}}(k_A)$.
- Node B verifies A's knowledge of k_M by verifying the image of n_B.

In short, the protocol looks like this:

$$1) A \rightarrow B : n_A$$

$$2) B \rightarrow A : n_B \,|\, h_{k^{j+1}_{auth}}(n_A) \,|\, e_{k^{j+1}_{auth}}(k_B) \,|\, j$$

$$3) A \rightarrow B : h_{k^{j}_{auth}}(n_B) \,|\, e_{k^{j+1}_{auth}}(k_A)$$

At this point, nodes A and B have successfully authenticated each other as well as exchanged their encryption keys. Once the fresh node considers all links established or a timer runs out, its own authenticator is generated and the master key replaced by the first authentication key.

Authenticator generation. Each pair is used only for one requester. Once a node runs out of the unused pairs, a new set ∇^{j+1} is generated. For this iteration a new set of random values is generated. Finally, the node computes a new authentication key k^{j+2}_{auth}, deletes the old one and increases the cycle counter j.

4 Analysis of Degado-Mohatar et al. Protocols

In this section, we describe the most problematic issues with the proposed protocols. Besides minor problems in the original paper such as wrong iterations of the key, there are some protruding security issues with the proposal even for the attacker model specified in this paper.

The attacker model for these Degado-Mohatar et al. protocols is only specified in [5] and not in [6]. They allow for a global active attacker prior to deployment with the possibility to capture a node. Obviously, that is an immensely strong attacker and as argued in [1], not completely realistic. However, the following discussion holds true even for our weaker attacker model.

4.1 Key Establishment Protocol Problems

Authors claim that each neighbouring pair of nodes would now share a *pairwise* encryption key. In fact, they share (at least) two keys, the one of node A as well as the one of node B. But these are not pairwise as pairwise means no other party shares this key. Still, node C hearing both n_A and n_B would have also computed the same encryption keys.

Furthermore, according to authors, their protocols provide *perfect resilience against node capture*, meaning that when compromising a node, only links that this node is involved in are compromised. However, in the proposed scheme, attackers successfully compromise an entire surrounding subnetwork and they have already some knowledge of the extent of such an attack.

Finally, there is no authentication of parties in this protocol, thus a resource exhaustion attack is easily executable by an unauthenticated node. In the current settings, a malicious node can impersonate multiple nodes during key establishment phase. While failing to successfully compute authenticated node's encryption key, it can severely deplete its memory by forcing it to compute and store large number of encryption keys.

4.2 Authentication Protocol Problems

Authors claim that the proposed protocol is designed for basically static networks with a minimum of new nodes joining the network. Therefore, the j in ∇^j should be always very small and a fresh node should be forced to compute only small number of hashes of the master key to receive k_{auth}^{j+1}.

However, even an unauthorized attacker Eve can reply to this fresh node's challenge with response :

$$E \rightarrow A : n_E, j,$$

where n_E is only sufficiently long and in the required format to mimic the expected message format

$$n_B, h_{k_{auth}^{j+1}}(n_A), e_{k_{auth}^{j+1}}(k_{enc}^B), j.$$

Then j can be set extremely high and thus force the fresh node to calculate an immense number of hashes, thus waste its battery before realizing that the data is nonsense.

The other issue with this protocol relates again to the node capture attack. Once an attacker captures a node and recovers an authentication key k_{auth}^j, he can authenticate new nodes against any current node in the m-th cycle with k_{auth}^m, where $m \geq j$. All he needs to be able to do is to synchronize the authentication cycle, i.e., hashing the acquired key $m - j$ times.

According to authors, a node capture attacker obtains only the authenticator of the current cycle and is therefore not able to compromise the authentication and exchange of keys performed using previous cycles. We stress that by capturing the node he captures the already stored keys with the authenticated nodes.

5 Proposed Protocols

In this section we give our suggestions for alternative protocols that are computationally efficient and have substantially better security properties.

One major difference to [6] is that we omit usage of the authentication operators as we consider them redundant. All the authentication information stored in them can be generated on the fly and thus lead to reductions in the storage needed as well as in the length of the messages exchanges.

5.1 Key Establishment Protocol, Proposal I

In this section we propose a key establishment protocol loosely based on [6], combined with key infection [1]. Our protocol provides *truly pairwise keys* for neighbouring nodes and achieves the security objectives.

In order to provide the functionality for future authentication of nodes joining the network, we establish also the authentication key.

Definition 3. *The* authentication key k_A *of node* A *is constructed as* $k_A = h(k_M|n_A)$.

Note that the nonce n_A is fresh, i.e., different to the one broadcasted during the pairwise key establishment.

The protocol can be described in these steps:

1. A common network-wide symmetric key k_M and an unique identifier is loaded to each node prior to deployment of the network.
2. After deployment, during the initialization phase each node broadcasts
$$A \to * : ID_A|e_{k_M}(ID_A|n_A),$$
where ID_A is used to differentiate from random messages early on.
3. Each node contains a set of similar messages from the surrounding nodes and can compute a new unique shared key with each of them as
$$k_{AB} = h(k_M|n_A|n_B).$$
4. At the end of the initialization phase, node A generates its authentication key k_A, saves n_A and deletes the master key k_M.

Note that A does not store the information needed to recover the link key of two neighbouring nodes B and C.

To authenticate a new node A after the initialization phase, it is loaded with the master key k_M. Node B is old and has only the authentication key. The protocol can be described in these steps:

1. Node A generates a random challenge n_A and sends it to B.
2. Node B replies with n_A concatenated with the material n_{KB} encrypted under k_B to generate the new pairwise key, and a nonce n_B used to generate its authentication key.

3. Node A computes the key k_B from n_B and k_M, verifies B's previous knowledge of k_M by decrypting $e_{k_B}(n_A|n_{KB})$ and verifying value of n_A, recovers the nonce n_{KB}, and responds with its part of the pairwise key n_{KA} concatenated with n_{KB} and encrypted under k_B as $e_{k_B}(n_{KA}|n_{KB})$. Node A then erases k_B.
4. Node B verifies the value n_{KB} to authenticate node A.
5. Both parties compute the new pairwise key $k_{AB} = h(n_{KA}|n_{KB})$.
6. Once the new node authenticates with each neighbour or the timer runs out, the node computes its own authentication key and erases k_M.

In short, the messages exchanged look like this:

$$1) A \rightarrow B : n_A$$

$$2) B \rightarrow A : e_{k_B}(n_A|n_{KB}) \,|\, n_B$$

$$3) A \rightarrow B : e_{k_B}(n_{KA}|n_{KB})$$

Once the new node authenticates with each neighbour or the timer runs out, the node computes its own authentication key and erases k_M.

Yet, as we realized later, the initialization phase part of the protocol is very similar to the BROSK scheme [9], with an improvement of slightly reducing the length of the message. The difference to BROSK is that instead of sending $ID_A|n_A|h_{k_M}(ID_A|n_A)$, we reduce the size of the message as it is the main parameter for the power consumption. However, this strongly depends on chosen algorithms and sizes of variables. Aditionally, in [9], the master key treatment after key establishment is missing. Thus the master key either stays in the nodes, which makes it vulnerable to a node capture attack or it gets deleted after some time and no delayed join to the network is possible.

Finally, it turns out that the entire proposal is almost identical to the one in LEAP protocol [16]. In LEAP, authors use BROSK scheme and follow it up with the same idea of using an "authentication key" of their own.

5.2 Key Establishment Protocol, Proposal II

Another alternative solution achieving given goals is built as a combination of the EG random key predistribution [8] and a preloaded network-wide key. In this protocol, we utilize random key predistribution as the primary method of pairwise key establishment. Additionally, we utilize the master key to improve the security of the EG scheme as well as for postprocessing to establish keys between nodes, where there has been no match of preloaded keys.

The protocol goes as follows:

1. Nodes broadcast preloaded key indexes encrypted under the master key.
2. Nodes establish pairwise keys with neighbours based on the shared keys.
3. Nodes establish pairwise keys with neighbours where there have been no shared keys found using the master key.

The addition of new nodes can be done in three ways, i.e., (a) the mechanism with local authentication keys as in 5.1; (b) the node can be loaded with another subset of the random key pool; and (c) a combination of (a) and (b).

6 Performance and Security Evaluation

In this section, we provide a discussion on the performance, from both memory and computational point of view. Also, we evaluate the security parameters of our proposals, especially in comparison to [6].

The memory requirements of our first proposal are significantly better to the ones provided in [6]. The results are summarized in table 1 for our proposal and table 2 for the previous work. The main difference here is the lack of the private key, addition of the nonce used to generate the authentication key and most importantly lack of the authentication operator.

In our settings, we use an 8-byte long nonce and a standard 16-byte long hash. Following the example from [6], in a network of 5 neighbours we come to 104 bytes of memory used for storage of keying material in comparison to the 352 bytes used in [6]. The second proposal's memory effeciency completely depends on the mechanism for future authentication chosen.

Table 1. Memory organization after initialization phase

Element	Notation	Size
Pairwise key with a neighbour	k_{AB}	16 bytes
Pairwise key with a neighbour	k_{AC}	16 bytes
	$\vdots$	
Authentication key	k_A	16 bytes
Nonce to generate auth. key	n_A	8 bytes

Table 2. Delgado-Mohatar et al. scheme's memory organization after initialization phase

Element	Notation	Size
Node encryption key	k_A	16 bytes
Neighbour encryption key	k_B	16 bytes
	$\vdots$	
First authentication key	k^1_{auth}	16 bytes
First cycle authenticator	∇^0	$x \times 24$ bytes

6.1 Proposal I

We provide authentication by encryption. Consequently, unauthorized nodes can not exhaust the receiver's memory.

Our messages are longer and encrypted over the [6] scheme. However, the encryption process brings only a small overhead and if the length of the nonce n_A is chosen carefully, we can end up with similar length of the encrypted data block, bringing again only a small overhead to the communication process.

In our settings, we use 2-byte long identifiers and the AES encryption with a 16 byte long block, thus resulting in an 18-byte long message. In [6], authors need only the 8-byte long nonce. Using the BROSK scheme, i.e., with MAC over encryption, the message length depends on the shortening of the MAC to receive required security. With the entire MAC being transported, 160 bit HMAC would result in a 30-byte long message.

Note that the original key infection paper [1] argues it unnecessary to encrypt this communication as an attacker who later recovers the master key anyway would be able to recover all previous communication intercepted. The attacker still needs some time to recover the key, for which all communication is secure, unlike for the case without encryption. Furthermore, an attacker might not even be able to recover the key, in which case we would unnecessarily disclose part of the communication. We allow for an active attacker, while in [1] no such attacker is allowed. This decision is based on the assumptions from [6].

An attacker capturing a node after the initialization phase is completed is able to recover only the actual keys used by that node. Therefore, all the other nodes such an attacker would try to add to the network would be authenticated only with the captured nodes.

If we omit the requirement of secured initialization phase for the newcoming node, we could further differentiate the master key used after network deployment. Instead of generating the authentication key directly from the original master key, we can first hash the master key then use it to construct our authentication key as

$$k_A = h(h(k_M), n_A).$$

In this manner, the newcoming nodes would contain only hash of the original master key. Although all the link keys established using this hashed key would be compromised, the original network built using the unhashed master key would keep all the link keys secret.

Similarly to [6], an attacker spamming challenges is dealt with using a simple timeout for each response. Thus, the implying DoS attack is no longer possible.

Our delayed authentication mechanism is also better with respect to the length of the messages exchanged over the authentication protocol from [6]. In our settings, the first message is just the same. The second message is 24 bytes long, while in [6] it is 45 bytes long. For the third message it is 16 and 36 bytes, respectively.

6.2 Proposal II

Scalability is an issue, but a managable one. However, the protocol provides good security properties based on the building blocks. The evaluation of the protocol depends on those. The EG scheme is a well known one. The case of delayed authentication done by EG scheme therefore as well. The case of our previous proposal is described in previous subsection.

7 Conclusions

We analyzed the scheme proposed in [6] and [5]. We have shown a number of its flaws and how they affect the security and energy efficiency of the entire network. We proposed simple attacks that make the network vulnerable even to an unauthorized attacker.

We presented two key establishment protocols, hopefully fixing all the problems found. These protocols allow for mutual key establishment during the initialization phase using a shared network-wide key and, after deletion of this key, authentication of new nodes joining the network.

We gave a performance and security analysis of our results, especially comparing it to [6]. In some ways like memory requirements or exchanged messages length, our protocols also exceed results from [6]. As for memory requirements, our proposal requires over 70% less memory when compared to the [6] given specified setup. Although the message broadcasted during key establishment is longer than the one in [6], it is authenticated. Finally, in the first proposal, we save up to 41 bytes in the length of messages exchanged.

Acknowledgment. We are grateful to Jiří Kůr for useful discussions and suggestions. Also, we thank the anonymous reviewers for their valuable comments and suggestions that improved the paper. This work was supported by the project GAP202/11/0422 and Vashek Matyas has undertaken most work on this paper as a Fulbright-Masaryk Visiting Scholar at Harvard University.

References

1. Anderson, R., Chan, H., Perrig, A.: Key infection: Smart trust for smart dust. In: Proceedings of the 12th IEEE International Conference on Network Protocols, pp. 206–215. IEEE Computer Society, Washington, DC (2004)
2. Blom, R.: An Optimal Class of Symmetric Key Generation Systems. In: Beth, T., Cot, N., Ingemarsson, I. (eds.) EUROCRYPT 1984. LNCS, vol. 209, pp. 335–338. Springer, Heidelberg (1985)
3. Chan, H., Perrig, A.: Pike: peer intermediaries for key establishment in sensor networks. In: Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2005, vol. 1, pp. 524–535 (March 2005)
4. Chan, H., Perrig, A., Song, D.: Random key predistribution schemes for sensor networks. In: 2003 Symposium on Security and Privacy, pp. 197–213 (May 2003)
5. Delgado-Mohatar, O., Fúster-Sabater, A., Sierra, J.M.: A light-weight authentication scheme for wireless sensor networks. Ad Hoc Networks 9(5), 727–735 (2011)
6. Delgado-Mohatar, O., Sierra, J.M., Brankovic, L., Fúster-Sabater, A.: An Energy-Efficient Symmetric Cryptography Based Authentication Scheme for Wireless Sensor Networks. In: Samarati, P., Tunstall, M., Posegga, J., Markantonakis, K., Sauveron, D. (eds.) WISTP 2010. LNCS, vol. 6033, pp. 332–339. Springer, Heidelberg (2010)
7. Du, W., Deng, J., Han, Y.S., Varshney, P.K., Katz, J., Khalili, A.: A pairwise key predistribution scheme for wireless sensor networks. ACM Trans. Inf. Syst. Secur. 8, 228–258 (2005)

8. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, pp. 41–47. ACM, New York (2002)
9. Lai, B.C.C., Hwang, D.D., Kim, S.P., Verbauwhede, I.: Reducing radio energy consumption of key management protocols for wireless sensor networks. In: Proceedings of the 2004 International Symposium on Low Power Electronics and Design, ISLPED 2004, pp. 351–356. ACM, New York (2004)
10. Lai, B., Kim, S., Verbauwhede, I.: Scalable session key construction protocol for wireless sensor networks. In: IEEE Workshop on Large Scale RealTime and Embedded Systems (LARTES), p. 7 (2002)
11. Lee, J.-Y., Stinson, D.R.: Deterministic Key Predistribution Schemes for Distributed Sensor Networks. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 294–307. Springer, Heidelberg (2004)
12. Perla, E., Catháin, A.O., Carbajo, R.S., Huggard, M., Mc Goldrick, C.: Powertossim z: realistic energy modelling for wireless sensor network environments. In: Proceedings of the 3nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, PM2HW2N 2008, pp. 35–42. ACM, New York (2008)
13. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: Spins: security protocols for sensor networks. Wirel. Netw. 8, 521–534 (2002)
14. Rasul, K., Nuerie, N., Pathan, A.: An enhanced tree-based key management scheme for secure communication in wireless sensor network. In: 2010 12th IEEE International Conference on High Performance Computing and Communications (HPCC), pp. 671–676 (September 2010)
15. Zhang, J., Varadharajan, V.: Wireless sensor network key management survey and taxonomy. Journal of Network and Computer Applications 33(2), 63–75 (2010)
16. Zhu, S., Setia, S., Jajodia, S.: Leap: efficient security mechanisms for large-scale distributed sensor networks. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, pp. 62–72. ACM, New York (2003)

Parameterized Algorithms for Stochastic Steiner Tree Problems

Denis Kurz, Petra Mutzel, and Bernd Zey

Department of Computer Science, TU Dortmund, Germany
{denis.kurz,petra.mutzel,bernd.zey}@tu-dortmund.de

Abstract. We consider the Steiner tree problem in graphs under uncertainty, the so-called two-stage stochastic Steiner tree problem (SSTP). The problem consists of two stages: In the first stage, we do not know which nodes need to be connected. Instead, we know costs at which we may buy edges, and a set of possible scenarios one of which will arise in the second stage. Each scenario consists of its own terminal set, a probability, and second-stage edge costs. We want to find a selection of first-stage edges and second-stage edges for each scenario that minimizes the expected costs and satisfies all connectivity requirements. We show that SSTP is in the class of fixed-parameter tractable problems (FPT), parameterized by the number of terminals. Additionally, we transfer our results to the directed and the prize-collecting variant of SSTP.

1 Introduction

The Steiner tree problem in graphs (STP) plays a central role in network design [14]. It asks for a minimum-cost subgraph of an undirected, weighted graph G that interconnects a given set of terminal nodes in G. It has applications in VLSI design [15,20], various communication systems, and often appears as a sub-problem of other network design problems [14].

The Steiner tree problem belongs to Karp's classical 21 NP-complete problems [16]. It is known to be NP-hard even if the input graph is unweighted and bipartite, i.e., containing only edges between terminal and non-terminal nodes [14]. Bern and Plassmann showed that the Steiner tree problem is Max-SNP-hard [1]. Therefore, there is no polynomial-time approximation scheme for STP. The best known constant-factor approximation was introduced by Byrka et al. [5] and guarantees an approximation factor of 1.39.

The most popular parameterized algorithm for STP is due to Dreyfus and Wagner [9]. In 1971, they introduced an algorithm that solves STP instances with n nodes, m edges, and t terminals in time $\mathcal{O}(3^t n^3)$, placing it in the complexity class FPT (for an introduction to the field of parameterized complexity see, e.g., [8,17]). Björklund et al. [3] were able to speed up the classical Dreyfus-Wagner algorithm to achieve a running time of $\mathcal{O}(2^t n^2 M + nm \log M)$ if all edge weights are in $\{1, \ldots, M\}$. At the same time, Fuchs et al. [10] published an algorithm with running time $(2+\delta)^t n^{\mathcal{O}(1/(\delta/\ln(1/\delta))^\zeta)}$ for any $\frac{1}{2} < \zeta \leq 1$ and sufficiently small $\delta > 0$. With respect to the graph's treewidth tw the best parameterized algorithm is due to Chimani et al. [7] and requires running time $\mathcal{O}(B_{tw+2}^2 \cdot tw \cdot n)$, where B_k is the k-th Bell number.

In the directed Steiner tree problem (DSTP), we are given a directed, weighted graph, a root node and a set of terminals. The objective is to find a minimum-cost subgraph that provides a directed path from the root node to each terminal. This problem is of theoretical interest as there are approximation-preserving reductions from many other problems to DSTP [6]. Unfortunately, there is no polylogarithmic approximation algorithm for DSTP unless NP $\subseteq$ ZTIME($n^{\text{polylog}(n)}$) [13].

In practice, we often have to face uncertainty. For example, a telecommunication company has to deal with volatile cable costs or an unpredictable set of customers. One approach to tackle uncertainty is (two-stage) stochastic optimization, cf. e.g. [2]. In the two-stage stochastic Steiner tree problem (SSTP), we do not know which terminal nodes have to be connected. Instead of buying edges to connect a given terminal set, we may buy edges that seem to be a good choice given a set of future scenarios one of which will eventually arise. We have to pay the current costs in this so-called first stage.

In a second stage, one of the scenarios is drawn at random, based on a previously known distribution. Each scenario is characterized by its terminal set, second-stage edge costs and a probability. We have to buy edges at second-stage costs to extend the first-stage edges to a Steiner tree for the scenario's terminal set. The *expected cost* of a solution is the sum of first-stage edge costs plus the weighted sum of second-stage edge costs of all scenarios. The weights correspond to the scenario's probability.

Obviously, the edges that have to be bought in the second stage depend on the edges bought in the first stage. We want to find a set of first-stage edges that minimizes the expected cost.

The first constant-factor approximation for SSTP was introduced by Gupta et al. [12]. Their approach requires the second-stage costs to be globally uniform over the different scenarios. They also require an inflation factor, i.e., a fixed ratio between first- and second-stage costs for all edges. Further approximation algorithms for the SSTP were provided by Gupta et al. [11] and by Swamy and Shmoys [19]. Chimani et al. [4] introduced an ILP-based exact algorithm for SSTP.

2 Definitions

A directed graph $G = (V, E)$ consists of a finite set V of nodes and a set $E \subset V \times V$ of edges. In an undirected graph, edges are considered to have no direction, and hence $(v, w) = (w, v)$.

A weighted graph $G = (V, E, c)$ is a graph with an additional edge cost function $c : E \mapsto \mathbb{R}_0^+$. It is symmetrical for undirected graphs, i.e., $c((v, w)) = c((w, v))$ for each $(v, w) \in E$.

The optimization problems defined below are minimization problems according to [18]. We characterize them by giving their set of instances, the set of feasible solutions for each instance, and a value function for selected elements.

The undirected and directed version of the Steiner tree problem differ slightly. We will focus on the undirected case first, which is also the most special variant.

Let $G = (V, E)$ be an undirected graph. A *Steiner tree* S for a *terminal set* $T \subseteq V$ in G is a connected subgraph of G that spans T. We allow S to contain non-terminal nodes of G. Such nodes are then called *Steiner nodes.* Note the difference between non-terminal and Steiner nodes: You can tell whether a node is a non-terminal node by just looking at the input. In contrast, a Steiner node is part of the solution. Every Steiner node is also a non-terminal node.

An instance of the *Steiner tree problem* (STP) consists of an undirected, weighted graph $G = (V, E, c)$ and a terminal set $T \subseteq V$. Feasible solutions are subsets of E that form the edge set of a Steiner tree for T in G. Edge values are their costs c. The cost $c(S)$ of S is the sum of costs of its edges.

The cost of a Steiner tree is used as a quality measure when considering Steiner optimization problems. Since it only depends on the edges, we will identify Steiner trees with their corresponding edge sets.

Now, let $G = (V, E)$ be a directed graph with designated root node $r \in V$ and $T \subseteq V$ a terminal set in G. A *Steiner arborescence* is a subgraph S of G such that for every terminal $t \in T$ there exists a directed path from r to t in S. As the name suggests, minimal Steiner arborescences are arborescences rooted in r. Steiner nodes and costs are defined analogously to the undirected case.

An instance of the *Steiner arborescence problem* (*directed Steiner tree problem*, DSTP) consists of a directed, weighted graph $G = (V, E, c)$ with designated root node $r \in V$, and a terminal set $T \subseteq V$. Feasible solutions are subsets of E that form the edge set of a directed Steiner tree for T in G rooted in r. Edge values are their costs c.

An instance of the *prize-collecting Steiner tree problem* (PCSTP) consists of an undirected, weighted graph $G = (V, E, c)$ and node profits $g : V \mapsto \mathbb{R}_0^+$. The set of nodes with positive profit is denoted by $T := \{v \in V \mid g(v) > 0\}$. Feasible solutions consist of a terminal set $\tilde{T} \subset T$ and a Steiner tree $F \subseteq E$ for $\tilde{T}$ in G. The value of a terminal $t \in T$ is given as $-g(t)$, while the value of an edge is its cost c. Therefore, the value of a solution is $\sum_{e \in F} c(e) - \sum_{t \in \tilde{T}} g(t)$.

A *stochastic Steiner tree* combines several Steiner trees for different terminal sets. Let $G = (V, E)$ be a graph, $r \in V$ a designated root node, $T^0 = \{r\}$, and $\mathcal{T} = \{T^k \mid 1 \leq k \leq K\}$ a set of K terminal sets. A stochastic Steiner tree for $\mathcal{T}$ in G rooted in r consists of first stage edges $F^0 \subseteq E$ and second stage edges $F^k \subseteq E$ for each scenario k that meet the following requirements. For each $0 \leq k \leq K$, there must exist a connected Steiner tree for T^k in G whose edge set is $F^0 \cup F^k$. Note that this requires the first stage edges to be connected and to include the root node if there are any first stage edges.

An instance of the *two-stage stochastic Steiner tree problem* (SSTP) consists of an undirected, weighted graph $G = (V, E, c^0)$ with designated root node $r \in V$, and a set of scenarios $\{(T^k, c^k, p^k) \mid 1 \leq k \leq K\}$, each consisting of a terminal set T^k, an edge-cost function c^k, and the scenario's probability p^k, satisfying $\sum_{k=1}^{K} p^k = 1$. Feasible solutions are stochastic Steiner trees for $\{T^k \mid 1 \leq k \leq K\}$ in G rooted in r. The value function v_1 is given as

$$v_1(e) := \begin{cases} c^0(e), & \text{if } e \in F^0 \\ p^k \cdot c^k(e), & \text{if } e \in F^k. \end{cases}$$

In other words, we ask for a stochastic Steiner tree $(F^0, \ldots, F^K)$ that minimizes the expected cost

$$\sum_{e \in F^0} c^0(e) + \sum_{k=1}^{K} p^k \sum_{e \in F^k} c^k(e).$$

Analogously to the SSTP, we define a stochastic variant of PCSTP. An instance of the *two-stage stochastic prize-collecting Steiner tree problem* (SPCSTP) consists of an undirected, weighted graph $G = (V, E, c^0)$ rooted in $r \in V$ and a set of K scenarios. Each scenario consists of a node profit function $g^k : V \mapsto \mathbb{R}_0^+$, edge costs c^k and a probability p^k with $\sum_{k=1}^{K} p^k = 1$. The set of nodes with positive profit in scenario k is denoted by $T^k := \{v \in V \mid g^k(v) > 0\}$. Feasible solutions consist of terminal sets $\mathcal{T} = \{\tilde{T}^k \subseteq T^k \mid 1 \leq k \leq K\}$ and a stochastic Steiner tree for $\mathcal{T}$ in G rooted in r. The value function v_2 is given as

$$v_2(x) := \begin{cases} v_1(x), & \text{if } x \in E \\ -(p^k \cdot g^k(x)), & \text{if } x \in \tilde{T}^k. \end{cases}$$

Intuitively, we do not force the solution to connect all terminals, but only the most profitable ones. Therefore, we subtract the profit of the nodes that are connected from the cost of the corresponding stochastic Steiner tree. Again, node profits g^k have to be weighted by p^k.

3 Solving SSTP

In the following section we describe an algorithm to solve the two-stage stochastic Steiner tree problem with a running time that is parameterized by the number of terminals summed up over the single scenarios. The approach is based on the algorithm by Dreyfus and Wagner [9], which can be described as follows:

The classical algorithm by Dreyfus and Wagner utilizes the method of dynamic programming: Solve a problem by formulating it recursively and solving the subproblems in increasing order. In fact, Dreyfus and Wagner realized the following recursive nature of the Steiner tree problem. Let S be an optimal Steiner tree for a given weighted, undirected graph $G = (V, E, c)$, and $p \in T \subseteq V, |T| \geq 3$, a terminal. There exists a *joining node* q in S and a partition (T_1, T_2) of T that meets the following requirement. The Steiner tree S can be split into three subtrees S_1, S_2, S_3, where S_1, S_2 are Steiner trees for $T_1 \cup \{q\}, T_2 \cup \{q\}$, respectively, and S_3 is a shortest path from p to q.

The Steiner tree problem is solved by populating a table M of values of subsolutions (p, T'). The subsolutions are solved in increasing order. For $|T'| = 1$, optimal Steiner trees connecting $T' \cup \{p\}$ are shortest paths. For $|T'| > 1$, a

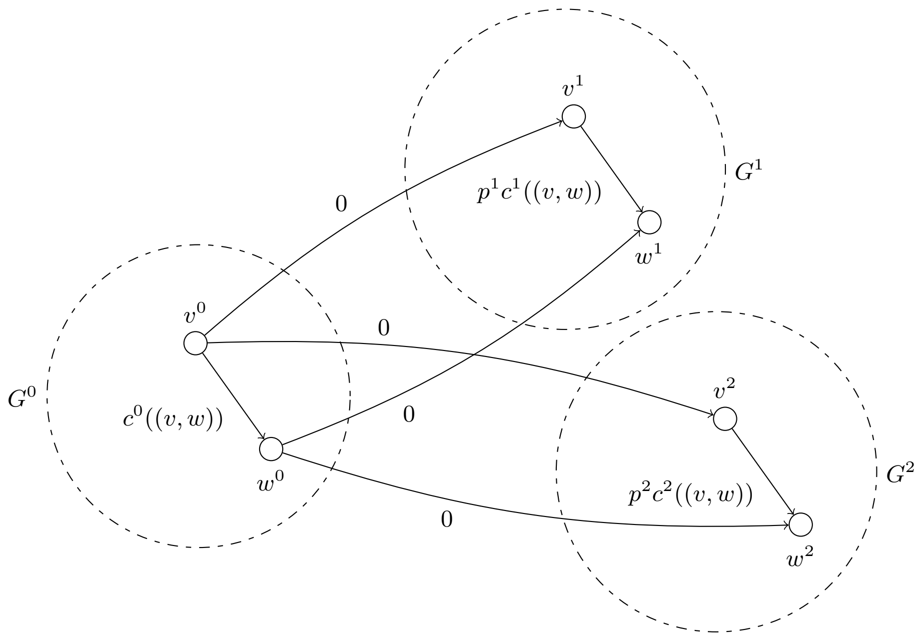


Fig. 1. Schematic diagram of the reduction from SSTP to DSTP with an example edge (v, w)

joining node and a partition of T is found by enumerating the nodes in V and the power set of T, respectively.

This approach can be carried over to the directed Steiner tree problem with designated root by constructing the optimal solution starting from the root. For a subproblem (p, T'), p serves as the root node.

We use these results to solve some stochastic versions of the Steiner tree problem. Let $G = (V, E, c)$ be a weighted, undirected graph with dedicated root node $r \in V$, and $\mathcal{T} := \{(T^k, c^k, p^k) \mid 1 \leq k \leq K\}$ a set of K scenarios. We construct an instance of the DSTP that has the same optimal value as the SSTP instance $(G, r, \mathcal{T})$. An optimal solution to the transformed instance also implies an optimal solution to the SSTP instance, and vice versa.

The directed graph $G' = (V', E', c')$ of this instance contains $K + 1$ copies of G, cf. Figure 1. Its node set V' is the union of all $V^k := \{v^k \mid v \in V\}$, $0 \leq k \leq K$. The edge set E' includes the corresponding edge sets of G, i.e., $E^k := \{(v^k, w^k), (w^k, v^k) \mid (v, w) \in E\}$ for $0 \leq k \leq K$.

Terminals of scenario k in G also become terminals in the new graph. Further, the root node becomes a terminal in the first-stage copy of G. This yields $T' = \{r^0\} \cup \{v^k \mid v \in T^k \setminus \{r\}, 1 \leq k \leq K\}$.

We will interpret the copies like this: Whenever we buy an edge (v^0, w^0) in the transformed version, we also buy this edge in the original SSTP instance, namely in the first stage. Accordingly, buying an edge (v^k, w^k), $k > 0$, suggests buying the corresponding edge in scenario k. We call (V^0, E^0) the first-stage copy and (V^k, E^k), $1 \leq k \leq K$, the second stage copy of the k-th scenario.

Now we have to make sure that there is a way to switch between the different copies of G in H. To accomplish this, we simply add edges between them. As two scenarios k and k' only interact in the first stage, it should not be allowed to switch from k to k' directly. Therefore, we only add *transition edges* between the first-stage copy on the one side and the various second stage copies on the other side. This yields $E' = \tilde{E} \cup \bigcup_{k=0}^{K} E^k$ with $\tilde{E} := \{(v^0, v^k) \mid v \in V, 1 \leq k \leq K\}$. Note that we only add edges from the first- to the second-stage copies, but not in the reverse direction.

The edge weights c' for the edges in E' are applied straightforwardly. Every edge weighs as much as its corresponding edge in E contributes to the value of the stochastic Steiner tree. Hence, first-stage edges remain unchanged, i.e., $c'((v^0, w^0)) = c((v, w))$ for $(v, w) \in E$. Second-stage edges have to regard the probability p^k of the scenario k they belong to, i.e., $c'((v^k, w^k)) = p^k \cdot c^k((v, w))$ for $1 \leq k \leq K$, $(v, w) \in E$. We do not want to restrict the number of transitions between first- and second-stage edges and therefore, we assign cost 0 to edges (v^0, v^k), for all $v \in V$.

We can now use the classical Dreyfus-Wagner algorithm to compute a solution S' to the DSTP instance (G', T'). This solution can be used to derive a solution $S = (F^0, \ldots, F^K)$ to the SSTP instance (G, r, T). We simply choose the edges that were also chosen in the corresponding copy of G in S'. Hence, we buy the edges $F^k := \{(v, w) \in E \mid (v^k, w^k) \in S'\}$ in the k-th scenario for $1 \leq k \leq K$, or in the first stage for $k = 0$.

We have to make sure that for each optimal solution to an SSTP instance there exists an equivalent solution to the transformed STP instance. Let $S = (F^0, \ldots, F^K)$ be a stochastic Steiner tree for $\{(T^k, c^k, p^k) \mid 1 \leq k \leq K\}$ in the undirected, weighted graph $G = (V, E, c^0)$. Let G' be the graph that results from the transformation from G as described before, and T' the corresponding terminal set.

As S is optimal, there is exactly one path $P(t) = (r = v_1, \ldots, v_\ell = t)$ from r to a terminal t of scenario k that only uses edges in $E^0 \cup E^k$. If this path uses first-stage edges, they are all grouped together at the beginning of the path. Otherwise, $P(t)$ would contain alternating fragments of first- and second-stage paths. But since the first stage is connected, this would induce a cycle, which contradicts our assumption that S is optimal. We denote by $\tau(t)$ the index of the transition from first- to second-stage edges, i.e., if $i < \tau(t)$ then $(v_i, v_{i+1}) \in P(t)$ is a first-stage edge and otherwise, $(v_i, v_{i+1}) \in P(t)$ is a second-stage edge.

The deterministic solution S' to (G', T') can be derived as follows. For each path $P(t) = (r = v_1, \ldots, v_\ell = t)$, $t \in T^k$, $1 \leq k \leq K$, we add the edges (v_i^0, v_{i+1}^0) from the first-stage copy to S' if $i < \tau(t)$. If $i \geq \tau(t)$, we add the edge (v_i^k, v_{i+1}^k) from the second-stage copy instead. Further, we add the required transition edge $(v_{\tau(t)}^0, v_{\tau(t)}^k)$.

For every edge e in S, we added an edge e' that contributes as much to the value of S' as e contributes to the value of S. The edges (v^0, v^k) do not contribute to the value of the solution at all. Therefore, the values of S and S' are equal.

In an analogous way an optimum solution to the DSTP instance can be transformed to an SSTP instance with the same objective value.

The new graph $G' = (V', E')$ has exactly $|V| \cdot (K + 1)$ nodes and $2 \cdot |E| \cdot (K + 1) + K \cdot |V|$ edges. We need to connect $t^* := \sum_{k=1}^{K} |T^k|$ terminals. Using the algorithm by Dreyfus and Wagner to solve the resulting DSTP instance, we obtain a running time of $\mathcal{O}(3^{t^*} \cdot (K \cdot |V|)^3)$. We summarize the previous results in the following theorem.

Theorem 1. *The stochastic Steiner tree problem is fixed parameter-tractable by the number of terminals. It can be solved in time $\mathcal{O}(3^{t^*} \cdot (K \cdot |V|)^3)$, where t^* is the sum of the number of terminals over all scenarios.*

Since DSTP is harder to approximate than STP, the question arises whether the reduction above would work in a similar way if the transition edges were undirected.

While transferring solutions from SSTP to a transformed (undirected) STP instance works without any problems, this is not true for the other way. Consider the STP solution that buys a minimum spanning tree of G with respect to c^1 and every undirected transition edge, i.e., $S' = \{(v^0, v^k) \mid v^0 \in V, 1 \le k \le K\} \cup \{(v^1, w^1) \mid (v^1, w^1) \in \mathrm{MST}(G, c^1)\}$. This solution is feasible for the constructed STP instance. The path from the root r to a terminal $t = v_\ell^k$ starts by switching to the second-stage copy of scenario 1 using the edge $(r = v_1^0, v_1^1)$. Following the unique path from v_1^1 to v_ℓ^1 provided by the minimum spanning tree, we reach v_ℓ^k by using (v_ℓ^1, v_ℓ^0) and (v_ℓ^0, v_ℓ^k).

In practice, this solution might even be a good candidate for an optimal solution. Edges in second-stage copies are weighted by a scenario's probability. Thus, they are often cheaper than their counterparts in the first-stage copy, especially if the probability of scenario 1 is very low.

However, transferring S' to the SSTP instance in a way analogous to the one above yields an infeasible SSTP solution S. It only contains edges in the second stage of scenario 1. The terminals of other scenarios remain unconnected.

4 Improvements

So far, we only considered solving SSTP by using the algorithm by Dreyfus and Wagner. This algorithm can be replaced by any other one that solves DSTP.

One way to improve the running time is to use the dynamic programming algorithm by Fuchs et al. [10]. It solves the Steiner tree problem for a terminal set $T \subseteq V$ in a weighted graph $G = (V, E, c)$. First, it adds a portion of $\frac{1}{\varepsilon}|T|$ terminals to the terminal set. An optimal Steiner tree for the new terminal set can then be split into optimal Steiner trees for at most $\varepsilon|T| + 1$ terminals. Obviously, this only works for a suitable selection of $\frac{1}{\varepsilon}|T|$ new terminals. Therefore, every possible selection is considered.

This approach yields an algorithm for STP with running time $(2 + \delta)^{|T|} n^{\mathcal{O}(1/(\delta/\ln(1/\delta))^{\zeta})}$ for any $\frac{1}{2} < \zeta \leq 1$ and sufficiently small $\delta > 0$ if we allow the Steiner tree to be constructed from parts of varying size. To utilize this algorithm to solve SSTP, we first need to show that it is capable of finding optimal directed Steiner trees.

Directed Steiner trees can be decomposed at inner terminals just like undirected Steiner trees. In contrast to the undirected STP, where sub-problems are fully characterized by their terminal sets, we also have to determine a root node within each terminal set for DSTP. Consider the directed Steiner tree problem for a terminal set $T \subseteq V$ in a weighted graph $G = (V, E, c)$ with designated root node $r \in T$, and a fixed partition $\mathcal{T}$ into subproblems. The following scheme provides us with a suitable choice of the root node for a given sub-problem with terminal set U.

Let $(\mathcal{T}_1, \ldots, \mathcal{T}_\ell)$ be a sequence of sub-problems with $\mathcal{T}_i \in \mathcal{T}$ for $1 \leq i \leq \ell$, $\mathcal{T}_i \cap \mathcal{T}_{i+1} \neq \emptyset$ and $\mathcal{T}_i \neq \mathcal{T}_\ell$ for $1 \leq i \leq \ell - 1$, $r \in \mathcal{T}_1$, and $\mathcal{T}_\ell = U$. In other words, the sequence is a path of sub-problems that starts in the sub-problem that contains r and ends in U, where adjacent sub-problems share at least one terminal.

If $\ell = 1$, we may choose r as the new root. Otherwise, there is exactly one node r' in the cut set $\mathcal{T}_{\ell-1} \cap \mathcal{T}_\ell$. If this cut set contained more than one node, combining the Steiner trees of all sub-problems would induce a cycle. This node r' has to be chosen as the root node for the sub-problem U.

Another way to speed up the computation of stochastic Steiner trees is to speed up the Dreyfus-Wagner algorithm itself. The fast subset convolution introduced by Björklund et al. [3] is one way to achieve this. A careful implementation of an underlying min-sum semiring allows them to compute an optimal Steiner tree in time $\tilde{\mathcal{O}}(2^k n^2 + nm)$ if edge weights are bounded by a constant, where $\tilde{\mathcal{O}}$ hides polylogarithmic factors.

Björklund et al. described how to apply their fast subset convolution to solve STP. However, they utilize the same recursive nature of the Steiner tree problem as Dreyfus and Wagner did. Therefore, the modifications made to the Dreyfus-Wagner algorithm to solve DSTP can also be applied to their approach.

5 Extensions

5.1 Directed SSTP

The reduction technique introduced in Section 3 can not only be used to solve the undirected SSTP. As the resulting graph is directed anyway, we might as well start with a directed graph. It is easy to see that a directed version of SSTP that asks for an arborescence for every scenario can be solved in much the same way as SSTP itself.

5.2 Prize-Collecting SSTP

To solve the prize-collecting version of the Steiner tree problem with the algorithm by Dreyfus and Wagner it needs some modifications. First, in contrast to the Steiner tree problems that connect terminals instead of collecting prizes, we do not have to connect every node of a given set. Instead, we may concentrate on the most profitable ones. Second, the value of a solution not only depends on the selected edges, but also on their incident nodes.

The first difference does not seem to be a problem at all. The Dreyfus-Wagner algorithm computes the value of every sub-problem, anyway. On this problem, every sub-solution that contains the root node is a feasible solution to the problem itself. Therefore, we have to memorize the value of the best solution we have seen so far. This value is only updated if the currently considered sub-problem connects the root node. Notice that it is very well possible that we get to see the optimal solution very early in the computation; This is not possible for terminal-connecting Steiner tree problems.

The latter difference requires some modifications to the Dreyfus-Wagner algorithm. We propose a modification that does not necessarily compute the correct values for every sub-solution. It does, however, compute correct values for the relevant subset of these sub-solutions.

The original Dreyfus-Wagner algorithm computes values $M(v, T')$ of sub-solutions, where v is an arbitrary node of the input graph and T' is a subset of its terminal set T. This value is computed as

$$M(v, T') = \min_{\substack{q \in V \\ T_1 \subset T \setminus \{v\}}} (d_G(v, q) + M(q, T_1) + M(q, T \setminus T_1)),$$

where d_G denotes distances in G.

To make this work for PCSTP, we need to consider the profit of connected nodes. Instead of detecting which profitable nodes with respect to a node-profit function g have been connected, we simply ignore the nodes that were included accidentally. In other words, we only consider profits of nodes that we intended to connect, but not those that were connected by default rather than by design. This yields the following recursive form to compute the values of sub-solutions:

$$M_{\text{PC}}(v, \tilde{T}) = \min_{\substack{q \in V \\ T_1 \cup T_2 = \tilde{T} \setminus \{q\}}} (M_{\text{PC}}(v, \{q\}) + M_{\text{PC}}(q, T_1) + M_{\text{PC}}(q, T_2)) + 2g(q))$$

The profit $g(q)$ of the joining node q has to be added twice because it would otherwise have been subtracted thrice, once for each M_{PC} that q is involved in.

The initialization of M has to be adjusted, too. Dreyfus and Wagner initialize $M(v, \{t\})$ for each node v and each terminal t. We also need $M_{\text{PC}}(v, \{q\})$ for arbitrary node pairs v, q. Further, we have to consider the node profits of the involved nodes. Therefore, we initialize M_{PC} with $M_{\text{PC}}(v, \{q\}) = d_G(v, q) - g(v) - g(q)$ for each $v, q \in V$, $v \neq q$, and $M_{\text{PC}}(v, \{v\}) = -g(v)$ for each $v \in V$.

Consider the value of $M_{\text{PC}}(e, \{a, b\})$ in the example in Figure 2. It is computed as the sum of three sub-solutions, minimized over all joining nodes and

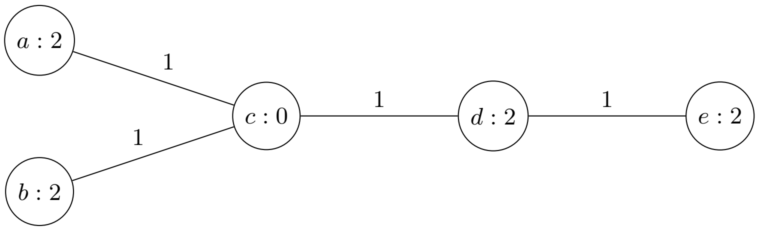


Fig. 2. PCSTP instance that includes sub-problems whose optimal values are not computed correctly; nodes are labelled with their name and their profit; edges are labelled with their cost

all partitions of $\{a, b\}$ with exactly two non-empty sets. Obviously, it can only be split into the sets $\{a\}$ and $\{b\}$. The sum of all sub-solutions for joining node q is $M_{\text{PC}}(e, \{q\}) + M_{\text{PC}}(q, \{a\}) + M_{\text{PC}}(q, \{b\}) + 2g(q)$. This sum evaluates to -2 or -3 for $q = c$ or $q = d$, respectively. In the first case, the edge (d, c) is used twice. In the second case, the profit of node d is ignored. In either case, the computed value is not optimal although the optimal selection of profitable nodes is connected.

However, the correct value of the optimal solution is still computed. We make sure that d is not ignored by including it in the set of nodes we want to connect. Although d is ignored when trying to connect e to $\{a, b\}$ without using it as a joining node, we do not ignore it when we try to connect e to $\{a, b, d\}$. In this case, the value of $M_{\text{PC}}(e, \{a, b, d\})$ would be computed as -4, which is the correct value of the optimal solution that connects all nodes in this example.

This observation can be formulated more general. Let $M_{\text{PC}}(v, \tilde{T})$ be a table entry that has not been computed correctly. The only way $M_{\text{PC}}(v, \tilde{T})$ can be incorrect is by ignoring profits of nodes that have actually been included. Let w be such a node. Then, the value of $M_{\text{PC}}(v, \tilde{T} \cup \{w\})$ is smaller than that of $M_{\text{PC}}(v, \tilde{T})$. This argument can be applied repeatedly until there are no ignored nodes left. The optimal set of profitable nodes is clearly found as we consider every subset of T.

These modifications also work for the directed case. They can thus be used in combination with the reduction technique from Section 3 to solve SPCSTP.

5.3 SSTP without a Root Node

One last extension allows us to provide not only one root for an SSTP instance, but a set of root candidates or no root at all. To allow for this, we do not add r^0 as a terminal during our reduction to the DSTP. Instead, we add a new node ρ that does not correspond to any node in the input graph G. This new node is then connected to the equivalent of each root candidate in the first-stage copy. These unidirectional edges are then equipped with high edge costs with the result that only one of the candidates is chosen.

These three extensions can be combined freely. It is therefore possible to solve, e.g., a directed stochastic prize-collecting Steiner tree problem using the classical algorithm and its modifications.

6 Conclusion and Outlook

We showed that SSTP is fixed-parameter tractable by the number of terminals. This result was subsequently extended to cover the directed and the prize-collecting SSTP.

It remains an open question whether SSTP and its variants can be parameterized by other parameters. One interesting parameter is the treewidth of the input graph. There exist parameterized algorithms for STP that utilize the treewidth. We have yet to investigate if these algorithms can be transferred to SSTP.

Another promising parameter is the number of non-terminals: A simple algorithm for STP tests every subset of non-terminals as the set of Steiner nodes and computes the corresponding minimum spanning tree. This approach might be transferable to the SSTP.

References

1. Bern, M.W., Plassmann, P.E.: The Steiner problem with edge lengths 1 and 2. Information Processing Letters 32(4), 171–176 (1989)
2. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer, New York (1997)
3. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: Fast subset convolution. In: STOC, pp. 67–74. ACM (2007)
4. Bomze, I., Chimani, M., Jünger, M., Ljubić, I., Mutzel, P., Zey, B.: Solving Two-Stage Stochastic Steiner Tree Problems by Two-Stage Branch-and-Cut. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part I. LNCS, vol. 6506, pp. 427–439. Springer, Heidelberg (2010)
5. Byrka, J., Grandoni, F., Rothvoß, T., Sanitá, L.: An improved LP-based approximation for Steiner tree. In: STOC, pp. 583–592. ACM (2010)
6. Charikar, M., Chekuri, C., Cheung, T., Dai, Z., Goel, A., Guha, S., Li, M.: Approximation algorithms for directed Steiner problems. In: SODA, pp. 192–200. SIAM (1998)
7. Chimani, M., Mutzel, P., Zey, B.: Improved Steiner Tree Algorithms for Bounded Treewidth. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2011. LNCS, vol. 7056, pp. 374–386. Springer, Heidelberg (2011)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
9. Dreyfus, S.E., Wagner, R.A.: The Steiner problem in graphs. Networks 1(3), 195–207 (1971)
10. Fuchs, B., Kern, W., Mölle, D., Richter, S., Rossmanith, P., Wang, X.: Dynamic programming for minimum Steiner trees. Theory Computing Systems 41(3), 493–500 (2007)
11. Gupta, A., Hajiaghayi, M.T., Kumar, A.: Stochastic Steiner Tree with Non-uniform Inflation. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX/RANDOM 2007. LNCS, vol. 4627, pp. 134–148. Springer, Heidelberg (2007)

12. Gupta, A., Pál, M., Ravi, R., Sinha, A.: Boosted sampling: Approximation algorithms for stochastic optimization. In: STOC, pp. 417–426. ACM (2004)
13. Halperin, E., Krauthgamer, R.: Polylogarithmic inapproximability. In: STOC, pp. 585–594. ACM (2003)
14. Hwang, F., Richards, D., Winter, P.: The Steiner tree problem. Annals of discrete mathematics, vol. 53. North-Holland (1992)
15. Kahng, A.B., Robins, G.: On Optimal Interconnections for VLSI. Kluwer Academic Publishers (1995)
16. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum (1972)
17. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Habilitation, Universität Tübingen (2002)
18. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Dover Publications (1998)
19. Swamy, C., Shmoys, D.B.: Approximation Algorithms for 2-Stage Stochastic Optimization Problems. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 5–19. Springer, Heidelberg (2006)
20. Uchoa, E., de Aragão, M.P., Ribeiro, C.C.: Preprocessing Steiner problems from VLSI layout. Networks 40(1), 38–50 (2002)

Action Investment Energy Games*

Kim G. Larsen, Simon Laursen, and Jiří Srba

Aalborg University, Department of Computer Science, Denmark
{kgl,simlau,srba}@cs.aau.dk

Abstract. We introduce the formalism of action investment energy games where we study the trade-off between investments limited by given budgets and resource constrained (energy) behavior of the underlying system. More specifically, we consider energy games extended with costs of enabling actions and fixed budgets for each player. We ask the question whether for any *Player 2* investment there exists a *Player 1* investment such that *Player 1* wins the resulting energy game. We study the action investment energy game for energy intervals with both upper and lower bounds, and with a lower bound only, and give a complexity results overview for the problem of deciding the winner in the game.

1 Introduction

Embedded systems are often executed on hardware with limited resources and they interact with uncontrollable or even hostile environments. By adding on top of this the ever increasing demand on software functionality and reliability for the lowest possible price, several interesting computational and optimization problems emerge. We introduce the formalism of action investment energy games where we study the trade-off between investments limited by given budgets and resource constrained (energy) behavior of the underlying system.

An action investment energy game (AIEG) is a two player game, played by *Player 1* and *Player 2*, each having a finite investment budget. The game consists of two independent phases: the *investment-configuration phase* and the *energy-game phase*. It is played on a finite multi-graph where transitions are labeled with action names and integers, representing energy changes. Furthermore each action has its own investment cost. In the investment-configuration phase, each player makes an investment by choosing a set of actions costing less than his/her budget. The result of these investments configures the board where an energy game is played on in the second energy-game phase. An energy game [3] is a turn based game, played on a finite multi-graph labeled with integer energy weights created in the investment-configuration phase. *Player 1* wins the energy game if she has a strategy such that the accumulated energy along any play according to the strategy is within a given interval.

Player 1 wins an AIEG if for any *Player 2* investment costing him less than his budget, it is possible for her to choose an investment costing less than her budget

* Supported by VKR Center of Excellence MT-LAB and the research center IDEA4CPS.

Player 1 actions	*coffee*	*coco*
Action cost	2	4

Player 2 actions	*MEMICS*	*CAV*
Action cost	4	6

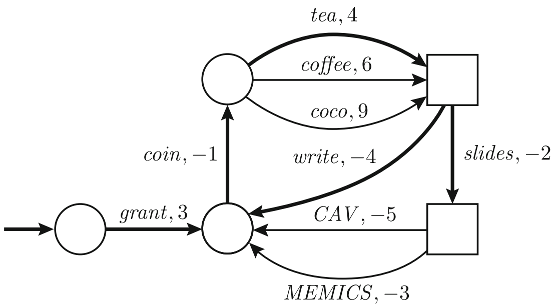


Fig. 1. Example of an action investment energy game

such that she wins the resulting energy game. Our focus is on the complexity of the decision problem asking whether *Player 1* has a winning strategy in the given AIEG.

An example of an action investment energy game is given in Figure 1. Here the nodes that have a circle shape belong to *Player 1* (she, young PhD student) and the squares are nodes that belong to *Player 2* (he, PhD supervisor who likes to stress PhD students to their limits). Every edge (transition) is labelled with the action name and a weight (energy change). The student repeatedly buys a drink of her choice after which she asks her adviser if she should keep writing a paper or prepare slides and present them at a conference. Having a drink increases her energy level while inserting a coin, writing a paper, preparation of slides and a trip to *CAV* or *MEMICS* cost energy. In order to survive, the student must always have a nonnegative energy level, while increasing the energy above a given upper-bound makes the student to quit. The thick transitions in the graph are always present while the thin ones can be enabled by the players in the first phase of the game, depending on their budgets. The student's budget gives her the opportunity to rent a drink-machine and configure the available drinks as long as they are within her budget. The supervisor has a travel budget allowing him to send the student to different conferences, provided it does not exceed the financial limit. In our example this means that *Player 1* can enable the actions *coffee* and *coco*, if her budget B_1 is sufficient for this and *Player 2* is in control of the actions *CAV* and *MEMICS*, relative to his budget B_2.

The game starts by *Player 2* investing in actions within his budget B_2. Then *Player 1* buys her actions up to the total cost B_1. All transitions under the selected actions are then included into the final graph. Now the players start moving a pebble across the graph such that *Player 1* selects her next move (edge) from any circle and *Player 2* from any square node. Starting with energy level 0, the energy weight on the selected edge is added to the so far accumulated energy. The question is whether for a given interval $[a, b]$, $a \leq 0 \leq b$ where b can be an infinity, *Player 1* has a winning strategy so that the accumulated energy stays within the interval $[a, b]$.

For example if the budget of *Player 1* is $B_1 = 4$ and the budget of *Player 2* is $B_2 = 5$, we can see that *Player 1* wins in the interval $[0, 9]$. The reason is that *Player 2* can choose only to invest in the action *MEMICS* as *CAV* is beyond the travel budget. *Player 1* then responds by buying the action *coffee* staying within her budget. The board of the game is now configured and the players start the energy-game phase. After getting a *grant*, inserting a *coin* and choosing the *coffee* transition, the student ends up in the situation with the accumulated energy equal to 8. Now *Player 2* can force the student to prepare the *slides* for a presentation and travel to *MEMICS*, reaching the accumulated energy 3 and returning the student to the configuration that already appeared (and hence it is winning for *Player 1*), or he can decide that the student should first write a paper, ending up with energy level 4. Now *Player 1* can insert a coin and choose to drink *coffee* again, increasing her accumulated energy to 9. Should *Player 2* decide to ask her to prepare slides and go to *MEMICS* now, the energy drops to 4. This is again a winning situation for *Player 1* as it has been reached before. If on the other hand *Player 2* decides that she should continue to work on the paper, the energy level reaches 5. Now *Player 1* can decide to drink a cup of *tea*, giving her the accumulated energy 8 that we have already seen before. Hence *Player 1* has a winning strategy in the game.

In fact the reader can verify that *Player 1* has a different winning strategy even in the interval $[0, 7]$ but if her budget does not allow to invest into neither *coffee* or *coco*, then *Player 1* losses for any given interval.

The main contribution of our work is the definition of action investment energy games and a detailed complexity analysis of the decision problem determining the winner of the game. In a work [1] related to ours, a similar trade-off scenario was studied where a dual-price schema for modal transition systems was introduced. Here the authors study the trade-off between a long-run average execution cost and a hardware investment cost, but they do not consider constrained resources and do not model uncontrollable environments. Several problems related to energy games were recently studied in [6,10,5,4], including extensions to real-time games [3,2] and imperfect information [9] but none of these works considered the investment phase. Another related formalism of feature transition systems [8,7] studies the problem of CTL/LTL model checking of the transition systems configured via a set of available features. However, features do not have any associated costs, the checked property is different from our energy condition and the game is restricted to 1-player only.

Proofs that are missing due to the space limitations can be found in the full version of the paper.

2 Definitions

We shall now present the definition of the action investment energy games. We start by recalling the notion of energy games.

2.1 Energy Game

Our notion of energy games is based on the definition from [3] where we consider general energy intervals $[a, b]$ instead of only $[0, b]$.

Definition 1 (Energy Game). *An* Energy Game *(EG) is defined as a tuple* $\mathcal{G} = (Q_1, Q_2, \Sigma, \rightarrow, q_0)$ *where*

- Q_1, Q_2 *are finite disjoint sets of states, we denote* $Q = Q_1 \uplus Q_2$,
- Σ *is a finite set of actions,*
- $\rightarrow \subseteq Q \times \Sigma \times \mathbb{Z} \times Q$ *is the successor relation where* $(q, \sigma, z, q') \in \rightarrow$ *is written as* $q \xrightarrow{\sigma,z} q'$, *and*
- $q_0 \in Q$ *is the initial state.*

An energy game can be depicted as a graph where each node represents a state such that circled nodes belong to Q_1 and squared ones to Q_2. Edges represent a transition between states and each edge is labeled with an action name and its weight. The energy game is played by moving a token around on the graph. The token starts in the initial state q_0. If the token is in a circle state, then *Player 1* moves the token to a successor state. Likewise, if the token is in a square state then *Player 2* moves the token to one of its successor states. The sequence on which the token moves is called a run, and it is defined as a finite or infinite sequence of transitions

$$r = q_0 \xrightarrow{\sigma_0,z_0} q_1 \xrightarrow{\sigma_1,z_1} q_2 \xrightarrow{\sigma_2,z_2} \ldots$$

where $q_i \in Q$ and $q_i \xrightarrow{\sigma_i,z_i} q_{i+1} \in \rightarrow$ for all i. Given a finite run $r = q_0 \xrightarrow{\sigma_0,z_0} \ldots \xrightarrow{\sigma_{n-1},z_{n-1}} q_n$, let the last state of the run be denoted by $Last(r) = q_n$. A run r is *maximal* if it is infinite, or finite and $Last(r)$ has no successors.

Definition 2 (Valid Run). *A maximal run* r *is valid in a given interval* $[a, b]$, $a \in \mathbb{Z}$, $b \in \mathbb{Z} \cup \{\infty\}$, $a \leq 0 \leq b$, *if* $a \leq \sum_{i=0}^{n} z_i \leq b$ *for all* n.

A play of the game can produce different runs depending on what strategy each player uses. A strategy δ_i for *Player i*, where $i = \{1, 2\}$, maps each finite non-maximal run r where $Last(r) = q_n \in Q_i$ to its successor $q_n \xrightarrow{\sigma_n,z_n} q_{n+1}$.

Problem 1 (Interval Bounded Energy Game). *Given an EG* $\mathcal{G}$ *and an interval* $[a, b]$, *does there exist a strategy* δ_1 *for Player 1 such that any play on* $\mathcal{G}$ *using the strategy* δ_1 *produces a run valid in the interval* $[a, b]$*?*

Player 1 wins and *Player 2* looses in the interval $[a, b]$ if the answer to Problem 1 in the interval $[a, b]$ is positive. There is also a more relaxed version of the interval bounded energy game called the *lower-bound problem* where the interval is of the form $[a, \infty]$.

2.2 Action Investment Energy Game

Let us split a given action alphabet Σ into three disjoint action sets $\Sigma_0 \uplus \Sigma_1 \uplus \Sigma_2 = \Sigma$ and call it the *action investment alphabet.* The action set Σ_i, $i = \{1, 2\}$, belongs to Player i, while Σ_0 is the set of default actions that are always present.

Definition 3 (Action Investment Energy Game). *An* action investment energy game *(AIEG) is a tuple,* $\mathcal{A} = (Q_1, Q_2, \Sigma, \rightarrow, q_0, actCost, B_1, B_2)$ *where,*

- $(Q_1, Q_2, \Sigma, \rightarrow, q_0)$ *is an energy game,*
- Σ *is an action investment alphabet,*
- $actCost : \Sigma_1 \cup \Sigma_2 \rightarrow \mathbb{N}$ *is a function assigning positive cost to the actions, and*
- $B_1, B_2 \in \mathbb{N}_0$ *are two nonnegative budgets.*

An *investment* is a subset of actions $I \subseteq \Sigma_1 \cup \Sigma_2$. The cost of an investment is the sum of the cost of the actions in the investment $invCost(I) = \sum_{\sigma \in I} actCost(\sigma)$. An investment for Player i, denoted by I_i, satisfies $I_i \subseteq \Sigma_i$.

Problem 2 (AIEG Problem). *Given an AIEG* $\mathcal{A}$ *and an interval* $[a, b]$*, does there for any Player 2 investment* $I_2 \subseteq \Sigma_2$ *where* $invCost(I_2) \leq B_2$ *exist a Player 1 investment* $I_1 \subseteq \Sigma_1$ *where* $invCost(I_1) \leq B_1$*, such that* Player 1 *wins the energy game* $\mathcal{G}' = (Q, Q_1, Q_2, \Sigma', \rightarrow', q_0)$ *in the interval* $[a, b]$*, where* $\Sigma' = \Sigma_0 \cup I_1 \cup I_2$ *and* $\rightarrow' = \rightarrow \cap (Q \times \Sigma' \times \mathbb{Z} \times Q)$*?*

The AIEG problem can be understood as a two-phase game: an *investment-configuration phase* and an *energy-game phase*. In the *investment-configuration phase*, *Player 2* starts by choosing his investment $I_2 \subseteq \Sigma_2$ costing less than his budget $invCost(I_2) \leq B_2$, then *Player 1* chooses her investment $I_1 \subseteq \Sigma_1$ costing less than her budget $invCost(I_1) \leq B_1$. This ends the first phase and the *energy-game phase* starts. In the *energy-game phase* the energy game is played on the reconfigured board where only actions in the two investments I_1, I_2 and the default actions from Σ_0 are present.

It is clear that for an AIEG problem where $\Sigma_1 = \emptyset$ and $\Sigma_2 = \emptyset$, or where $B_1 = 0$ and $B_2 = 0$, the AIEG problem reduces to the classical energy game as none of the players can make any investment.

3 Gadgets for Complexity Bounds

Our main contribution is a detailed complexity analysis of the general action investment energy game problem and some of its prominent subclasses. For this reason, we start by establishing several gadgets (basically instances of AIEG) that will be used in the next section in order to prove complexity lower-bounds by reductions from different variants of quantified boolean formulae satisfiability problem. Let us assume a set of n boolean variables $x_1, \ldots, x_n$ for which we consider the action alphabet $\{x_j, x'_j \mid 1 \leq j \leq n\}$. We start by the definition of a valid investment.

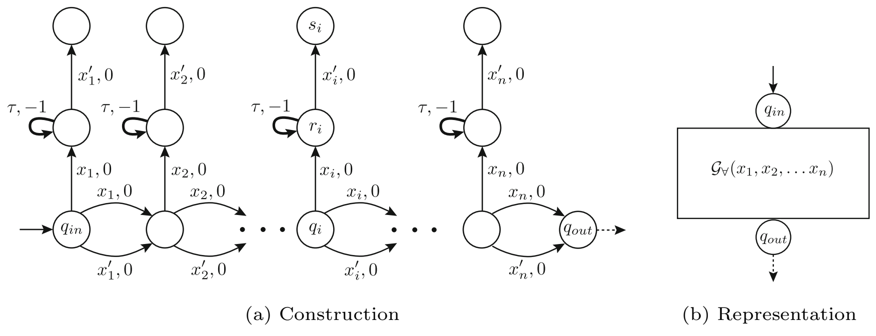


Fig. 2. Gadget $\mathcal{G}_\forall(\boldsymbol{x})$ where $\boldsymbol{x} = (x_1, \ldots, x_n)$

Definition 4 (Valid Investment). *Let $i \in \{1, 2\}$ and let $\Sigma_i = \{x_j, x'_j \mid 1 \leq j \leq n\}$. An investment $I_i \subseteq \Sigma_i$ is* valid *for Player i if for all j, $1 \leq j \leq n$, either $x_j \in I_i$ or $x'_j \in I_i$ and $\{x_j, x'_j\} \nsubseteq I_i$.*

Hence a valid investment represents an assignment of truth values to the boolean variables such that if $x_j \in I_i$ then the value of x_j is true and if $x'_j \in I_i$ then x_j takes the value false. It is clear that each player needs a sufficient budget to make a valid investment. This is defined in the next definition.

Definition 5 (Sufficient Budget). *A budget B_i is sufficient for Player i, where $i = \{1, 2\}$, if actCost$(I_i) \leq B_i$ for any valid investments $I_i \subseteq \Sigma_i$.*

3.1 Gadget $\mathcal{G}_\forall(\boldsymbol{x})$

The gadget $\mathcal{G}_\forall$ is used by *Player 2* to enforce a valid truth assignment of the variables $\{x_1, \ldots, x_n\}$ of his choice. In this gadget we let $\Sigma_2 = \{x_1, \ldots x_n, x'_1 \ldots x'_n\}$, and all states belong to *Player 1*. Let $\tau \in \Sigma_0$ be a default action that is always present. The construction of $\mathcal{G}_\forall$ and its graphical representation is given in Figure 2. The construction ensures that *Player 2* needs to make a valid investment otherwise *Player 1* has a strategy to win. In addition if *Player 2* chooses a valid investment then any play starting from q_{in} reaches q_{out} or it is loosing for *Player 1*. The following lemma formalizes this fact, assuming that *Player 2* has a sufficient budget to make a valid investment.

Lemma 1 (Properties of the Gadget $\mathcal{G}_\forall$)

(a) If I_2 is a valid investment for Player 2, then any play starting from q_{in} is either loosing for Player 1 or reaches q_{out} and Player 1 has moreover a strategy to ensure that any play from q_{in} reaches q_{out}.
(b) If I_2 is not a valid investment for Player 2, then Player 1 has a strategy to win any play starting from q_{in}.

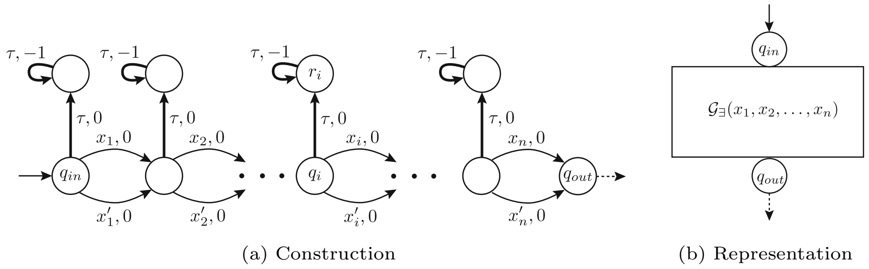


Fig. 3. Gadget $\mathcal{G}_\exists(\boldsymbol{x})$ where $\boldsymbol{x} = (x_1, \ldots, x_n)$

3.2 Gadget $\mathcal{G}_\exists(\boldsymbol{x})$

The gadget $\mathcal{G}_\exists(\boldsymbol{x})$ is used by *Player 1* to fix her truth assignment of variables $\{x_1, \ldots, x_n\}$. We let $\Sigma_1 = \{x_1, \ldots x_n, x'_1 \ldots x'_n\}$ and fix the budget for *Player 1* to correspond to the number of variables, in other words $B_1 = n$. We also assume that the cost of any action in Σ_1 is equal to 1 so that it is possible for *Player 1* to make a valid investment. Let $\tau \in \Sigma_0$ be a default action that is always present. The gadget and its graphical representation is depicted in Figure 3. The point is that *Player 1* needs to choose a valid investment, otherwise *Player 2* has a winning strategy. Note that all nodes in the gadget belong to *Player 1*, so she is the only one that decides the moves in this gadget.

Lemma 2 (Properties of the Gadget $\mathcal{G}_\exists$)

(a) If I_1 is a valid investment for Player 1, then any play starting from q_{in} is either loosing for Player 1 or reaches q_{out} and Player 1 has moreover a strategy to ensure that any play from q_{in} reaches q_{out}.
(b) If I_1 is not a valid investment for Player 1, then any play from q_{in} is winning for Player 2.

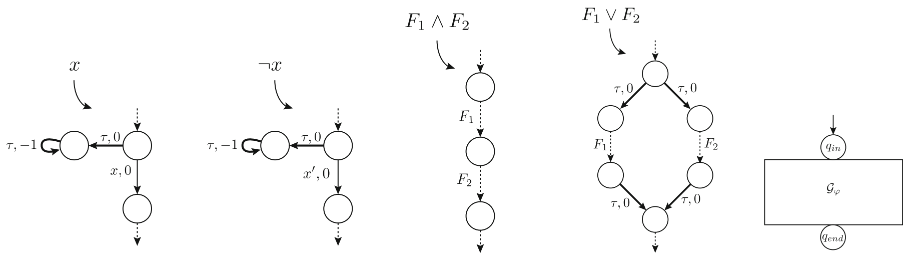
3.3 Gadget $\mathcal{G}_\varphi$

After introducing gadgets that allow the players of the AIEG game to choose valid investments that correspond to truth assignments of boolean variables, we need to make a gadget that will check if a boolean formula φ is true under the selected assignment. Such a gadget $\mathcal{G}_\varphi$ is defined inductively in Figure 4 where the dotted lines represents subgadgets already constructed for the subformulae.

For the proof of correctness, recall that $\Sigma_1 \cap \Sigma_2 = \emptyset$ and that $\tau \in \Sigma_0$ is a default action always present in the game.

Lemma 3 (Properties of the Gadget $\mathcal{G}_\varphi$)
Let I be a valid investment and let

$$v(x) = \begin{cases} true & \text{if } x \in I \\ false & \text{if } x' \in I \end{cases}$$



(a) Positive literal (b) Negative literal (c) Conjunction (d) Disjunction (e) Representation

Fig. 4. Gadget $\mathcal{G}_\varphi$

be the corresponding truth assignment.

(a) If φ is true under the assignment v then Player 1 has a strategy to win any play starting from q_{in} in $\mathcal{G}_\varphi$.
(b) If φ is false under the assignment v then Player 2 has a strategy to win any play starting from q_{in} in $\mathcal{G}_\varphi$.

3.4 Linking Gadgets

For the purpose of our complexity proofs in the following section, gadgets can be linked together in a sequence and thereby create a combined AIEG. This is done via a transition labelled with $\tau, 0$ starting from q_{out} in one gadget and leading to the state q_{in} of the other gadget. The shorthand notation for linking gadgets is an arrow such that for example $\mathcal{G}_\forall(\boldsymbol{x}) \to \mathcal{G}_\exists(\boldsymbol{y}) \to \mathcal{G}_\varphi$ corresponds to an AIEG starting with the universal gadget over the vector of variables $\boldsymbol{x}$, followed by the existential gadget over the vector of variables $\boldsymbol{y}$ and finished by the gadget checking the validity of a formula under the generated truth assignment. It is necessary to rename the states of the linked gadgets to avoid name clashes and to update the successor relation accordingly. This can be done in the expected way, so we omit the details here.

4 Complexity Results

We shall now present an overview of complexity results for different subclasses of the AIEG problem. The new results in Table 1 are listed in bold font and we consider the action investment energy game problem as well as its well-studied existential variant where all states belong to *Player 1*. We moreover study the variants of the game where one of the players has a zero budget (the case when both of them have zero budget corresponds to classical energy games) and we distinguish intervals that are closed or open to the right.

Some of the bounds in Table 1 use complexity classes from the polynomial hierarchy. We refer the reader to some classical textbook like [11] for more details

Table 1. Complexity Overview for AIEG Problems

Budget restrictions	Interval	Energy game type	
		Existential	Game
$B_1 = 0$, $B_2 = 0$	$[a, \infty]$	$\in$ P [3]	$\in$ UP $\cap$ coUP [3]
	$[a, b]$	NP-Hard, $\in$ PSPACE [3]	EXPTIME-complete [3]
$B_2 = 0$	$[a, \infty]$	**NP-Complete**, Lem. 4	**NP-Complete** Lem. 10
	$[a, b]$	**NP-Hard,** $\in$ **PSPACE**, Lem. 5	**EXPTIME-complete** Lem. 13
$B_1 = 0$	$[a, \infty]$	**co-NP-complete**, Lem. 6	**co-NP-Complete** 11
	$[a, b]$	Π_2^P**-hard**, $\in$ **PSPACE**, Lem. 7	**EXPTIME-complete** Lem. 13
–	$[a, \infty]$	Π_2^P**-complete**, Lem. 8	Π_2^P**-complete** 12
	$[a, b]$	Π_2^P**-hard** $\in$ **PSPACE**, Lem. 9	**EXPTIME-complete** Lem. 13

about the hierarchy. Most of the complexity bounds are a direct application of our gadgets presented in the previous section, apart from Lemma 7 that is considerably more involved as an additional binary encoding of multiple weights into a single integer is needed there.

Lemma 4. *The AIEG problem for the interval* $[a, \infty]$ *where* $Q_2 = \emptyset$ *and* $B_2 = 0$ *is NP-complete.*

Proof (sketch). For the lower bound let $\exists \boldsymbol{x} \varphi(\boldsymbol{x})$ be an instance of the NP-complete SAT problem over the vector of variables $\boldsymbol{x} = (x_1, x_2 \ldots x_n)$. We construct in polynomial time the AIEG $\mathcal{A}$ given by $\mathcal{G}_\exists(\boldsymbol{x}) \to \mathcal{G}_\varphi$ and fix the budget for *Player 1* to $B_1 = n$. It is now clear that $\exists \boldsymbol{x} \varphi(\boldsymbol{x})$ is true iff *Player 1* is the winner of the AIEG problem $\mathcal{A}$. For the upper bound, an algorithm for solving the problem guesses a *Player 1* investment and solves in polynomial time [3] the interval bound problem of the resulting energy game. □

Lemma 5. *The AIEG problem for the interval* $[a, b]$ *where* $Q_2 = \emptyset$ *and* $B_2 = 0$ *is NP-Hard and in PSPACE.*

Lemma 6. *The AIEG problem for the interval* $[a, \infty]$ *where* $Q_2 = \emptyset$ *and* $B_1 = 0$ *is co-NP-complete.*

Proof (Sketch). For the lower bound let $\forall \boldsymbol{x} \varphi(\boldsymbol{x})$ be an instance of Π_1^P-SAT (co-NP complete problem). We construct in polynomial time the AIEG $\mathcal{A}$ given by $\mathcal{G}_\forall(\boldsymbol{x}) \to \mathcal{G}_\varphi$ and let B_2 be any sufficient budget. It is now easy to see that $\forall \boldsymbol{x} \varphi(\boldsymbol{x})$ is true iff *Player 1* is the winner in the AIEG problem $\mathcal{A}$. An algorithm for solving the problem in co-NP enumerates using universal branching all possible *Player 2* investments and for each investment solves in polynomial time [3] the interval bound problem of the resulting energy game. □

Lemma 7. *The AIEG problem for the interval $[a, b]$ where $Q_2 = \emptyset$ and $B_1 = 0$ is Π_2^P-hard and in PSPACE.*

Proof (sketch). For the lower bound let $\forall \boldsymbol{x} \exists \boldsymbol{y} \varphi(\boldsymbol{x}, \boldsymbol{y})$ be an instance of Π_2^P-SAT. We construct an AIEG $\mathcal{A}$, illustrated on Figure 5. The first part of the construction is the gadget $\mathcal{G}_\forall(\boldsymbol{x})$ that ensures that *Player 2* is forced to select a valid investment over the variables $\{x_1, x'_1, \ldots, x_n, x'_n\}$ (assuming we select the budget B_2 as a sufficient one). It would be intuitive to link this gadget with the gadget $\mathcal{G}_\exists(\boldsymbol{y})$ to force *Player 1* to choose her valid investment over the variables $\{y_1, y'_1, \ldots, y_n, y'_n\}$, however, $B_1 = 0$ and *Player 1* can not make any

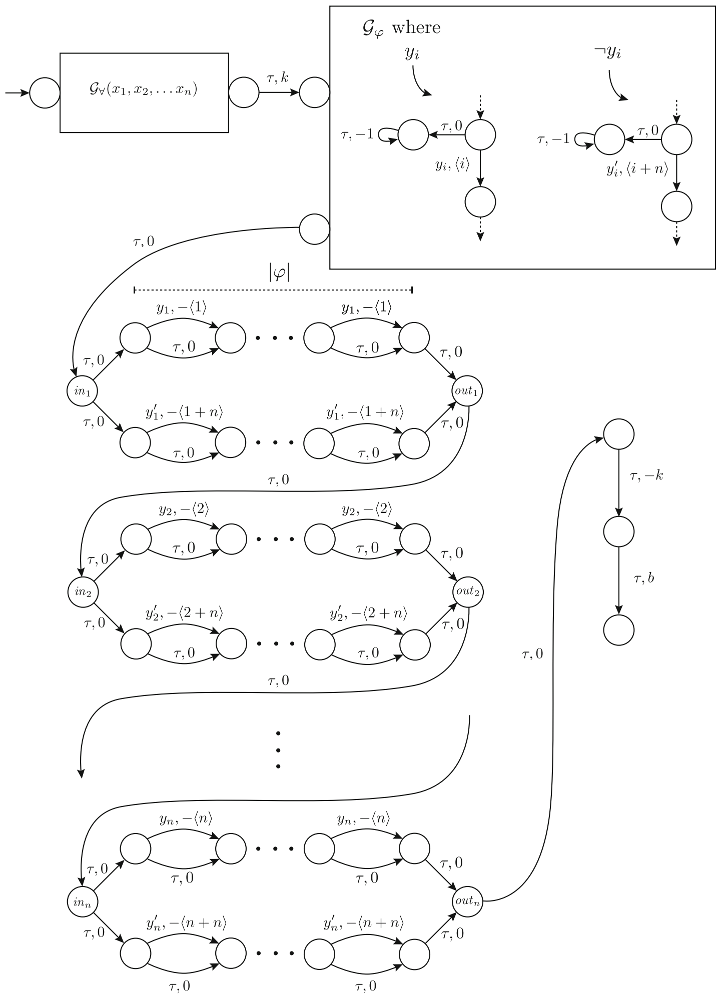

$\mathcal{G}_\forall(x_1, x_2, \ldots x_n)$
τ, k
$\mathcal{G}_\varphi$ where
y_i
$\neg y_i$
$\tau, -1$
$\tau, 0$
$y_i, \langle i \rangle$
$y'_i, \langle i+n \rangle$
$|\varphi|$
$y_1, -\langle 1 \rangle$
$y'_1, -\langle 1+n \rangle$
$y_2, -\langle 2 \rangle$
$y'_2, -\langle 2+n \rangle$
$y_n, -\langle n \rangle$
$y'_n, -\langle n+n \rangle$
in_1
out_1
in_2
out_2
in_n
out_n
$\tau, -k$
τ, b

Fig. 5. AIEG used in proof of Lemma 7

investment. Therefore we need to make an alternative construction. We shall use $2n$ counters in order to record how many times the actions y_i and y'_i were seen while traversing the modified gadget G_φ. The counters are encoded in binary, and the number of bits needed for each counter is given by $c = \lceil \log(|\varphi| + 1) \rceil$ where $|\varphi|$ is the number of literals that appear in the boolean formula φ. We need two counters for each boolean variable y_i such that the counter i counts the number of positive occurrences of the variable y_i and the counter $i+n$ counts the number of negative occurrences. All counters will be stored in a single integer. To ensure that counters do not get 'entangled' by under- or overflow, we add two separator bits 10 between any two neighboring counters, giving us that in total we need $2n(c+2)$ bits. We define $k = \sum_{i=1}^{2n} 2^{i(2+c)}$ as the initial counter value where all counter are 0 and with the separator bits between them. For an example if $n = 2$ and $c = 3$ then k is defined (in binary notation) as follows:

$$k = \overbrace{10\ \underbrace{000}_{\text{counter 4}}\ 10\ \underbrace{000}_{\text{counter 3}}\ 10\ \underbrace{000}_{\text{counter 2}}\ 10\ \underbrace{000}_{\text{counter 1}}}^{\text{20 bits}}.$$

Incrementing and decrementing of counters is done via weights on transitions, therefore we need a way to address each counter. Counter ℓ, $1 \le \ell \le 2n$, is addressed by $\langle \ell \rangle$ where $\langle \ell \rangle = 2^{(2+c)(\ell-1)}$. In this way if a transition with weight $\langle \ell \rangle$ is taken, the counter ℓ is incremented by one, and similarly when a transition with weight $-\langle \ell \rangle$ is taken, the counter ℓ is decremented by one. We set $b = 2^{2n(c+2)+1} - 1$ which is the highest possible number on $2n(c+2)$ bits (where all bits are 1) and will consider the resulted energy game in the interval $[0, b]$.

Now we can construct an alternative $\mathcal{G}_\varphi$ which is linked after the $\mathcal{G}_\forall(\boldsymbol{x})$ gadget as illustrated in Figure 5. The alternative $\mathcal{G}_\varphi$ gadget is as defined before, with the exception that every y_i literal gives rise to the $y_i, \langle i \rangle$ transition, and every $\neg y_i$ literal gives gives rise to $y'_i, \langle i + n \rangle$ transition, while assuming that $\{y_1, y'_1, \dots y_n, y'_n\} \subseteq \Sigma_0$. By taking a path from the start to the end state of this gadget, we will record in the counters how many times each literal has been seen on such a path. The initial $\mathcal{G}_\forall(\boldsymbol{x})$ gadget is linked via a τ, k transition to this new $\mathcal{G}_\varphi$ gadget, ensuring that all counters are initialized to 0 by adding only the separator bits. At the end of the construction, we add a series of gadgets for each variable y_i, allowing *Player 1* to decrement at most $|\varphi|$-many times either the counter i or $n + i$ but not both at the same time. If the assignment recorded in the counters is a valid one, *Player 1* can decrease all counter values to zero, such that after removing the separator bits by the transition $\tau, -k$ we get the accumulated weight 0 and it is possible to add the upper bound b without violating the energy game interval $[0, b]$. It can now be shown that $\forall \boldsymbol{x} \exists \boldsymbol{y} \varphi(\boldsymbol{x}, \boldsymbol{y})$ is true iff *Player 1* is the winner of the AIEG problem $\mathcal{A}$ for the interval $[0, b]$.

For the upper bound, an algorithm for solving the problem in PSPACE enumerates all possible *Player 2* investments and for each investment solves the interval bound problem of the resulting energy game. The interval bound problem in the existential case, $Q_2 = \emptyset$, for an interval $[a, b]$ is in PSPACE, implying that the problem remains in PSPACE. $\square$

Lemma 8. *The AIEG problem for the interval $[a, \infty]$ where $Q_2 = \emptyset$ is Π_2^P-complete.*

Proof (sketch). For the lower bound let $\forall \boldsymbol{x} \exists \boldsymbol{y} \varphi(\boldsymbol{x}, \boldsymbol{y})$ be a Π_2^P-complete instance of QSAT where $\boldsymbol{x} = (x_1, x_2 \ldots x_n)$ and $\boldsymbol{y} = (y_1, y_2 \ldots y_m)$. We construct the AIEG $\mathcal{A}$ given by $\mathcal{G}_\forall(\boldsymbol{x}) \rightarrow \mathcal{G}_\exists(\boldsymbol{y}) \rightarrow \mathcal{G}_\varphi(\varphi)$ and fix the budget $B_1 = n$ and let B_2 be any sufficient budget for *Player 2*. It is now easy to see that $\forall \boldsymbol{x} \exists \boldsymbol{y} \varphi(\boldsymbol{x}, \boldsymbol{y})$ is true iff *Player 1* is the winner of the AIEG problem $\mathcal{A}$. An algorithm for solving the problem enumerates via universal branching all possible *Player 2* investments and for each such investment guesses a *Player 1* investment and solves in polynomial time the resulting energy game. □

Lemma 9. *The AIEG problem for the interval $[a, b]$ where $Q_2 = \emptyset$ is Π_2^P-hard and in PSPACE.*

Lemma 10. *The AIEG problem for the interval $[a, \infty]$ where $B_2 = 0$ is NP-complete.*

Proof. The lower bound follows from Lemma 4. An algorithm for solving the problem guesses a *Player 1* investment and solves the interval bound problem of the resulting energy game which is in UP ∩ coUP, implying that the problem is in NP. □

Lemma 11. *The AIEG problem for the interval $[a, \infty]$ where $B_1 = 0$ is co-NP-complete.*

Proof. The lower bound follows from Lemma 6. An algorithm for solving the problem enumerates via universal branching all possible *Player 2* investments and for each investment solves the interval bound problem of the resulting energy game which is in UP ∩ coUP, implying that the problem is in co-NP. □

Lemma 12. *The AIEG problem for the interval $[a, \infty]$ is Π_2^P-complete.*

Proof. The lower bound follows from Lemma 8. An algorithm for solving the problem enumerates via universal branching all possible *Player 2* investments and for each such investment guesses a *Player 1* investment and solves the interval bound problem of the resulting energy game. The interval bound problem for an open interval $[a, \infty]$ is in UP ∩ coUP, implying that the problem is in Π_2^P. □

Lemma 13. *The AIEG problem for an interval $[a, b]$ is EXPTIME-complete.*

5 Conclusion

We have provided a complexity characterization of action investment energy games. The problem combines the action-investment phase with the energy-game phase and for many cases we proved matching complexity lower and upper bounds. Thanks to the general definition of our gadgets that can be combined using linking into different variants of the game, we were able to give intuitive constructions for most of the complexity results. A notable result is that for the interval problems with lower and upper bound, apart from the case where both budgets are zero, the complexity of the existential case and the general game setting remain the same. The few problems where we did not close the complexity bounds depend on the open problem of the existential energy game in the interval $[a, b]$, which is so far only known to be between NP and PSPACE.

We studied a version of AIEG where *Player 2* chooses his investment before *Player 1*. It is natural to consider also the opposite order in which the investment is established or even a turn-based investment phase. This will be studied in our future work.

References

1. Beneš, N., Křetínský, J., Guldstrand Larsen, K., Møller, M.H., Srba, J.: Dual-Priced Modal Transition Systems with Time Durations. In: Bjørner, N., Voronkov, A. (eds.) LPAR-18 2012. LNCS, vol. 7180, pp. 122–137. Springer, Heidelberg (2012)
2. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N.: Timed automata with observers under energy constraints. In: HSCC 2010, pp. 61–70 (2010)
3. Bouyer, P., Fahrenberg, U., Larsen, K.G., Markey, N., Srba, J.: Infinite Runs in Weighted Timed Automata with Energy Constraints. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 33–47. Springer, Heidelberg (2008)
4. Chatterjee, K., Doyen, L.: Energy Parity Games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 599–610. Springer, Heidelberg (2010)
5. Chatterjee, K., Doyen, L.: Energy and Mean-Payoff Parity Markov Decision Processes. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 206–218. Springer, Heidelberg (2011)
6. Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Generalized mean-payoff and energy games. In: FSTTCS. LIPIcs, vol. 8, pp. 505–516. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
7. Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A.: Symbolic model checking of software product lines. In: ICSE, pp. 321–330. ACM (2011)
8. Classen, A., Heymans, P., Schobbens, P.-Y., Legay, A., Raskin, J.-F.: Model checking lots of systems: efficient verification of temporal properties in software product lines. In: ICSE (1), pp. 335–344. ACM (2010)
9. Degorre, A., Doyen, L., Gentilini, R., Raskin, J.-F., Toruńczyk, S.: Energy and Mean-Payoff Games with Imperfect Information. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 260–274. Springer, Heidelberg (2010)
10. Fahrenberg, U., Juhl, L., Larsen, K.G., Srba, J.: Energy Games in Multiweighted Automata. In: Cerone, A., Pihlajasaari, P. (eds.) ICTAC 2011. LNCS, vol. 6916, pp. 95–115. Springer, Heidelberg (2011)
11. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)

Ciphertext-Only Attack on Gentry-Halevi Implementation of Somewhat Homomorphic Scheme*

Michal Mikuš and Marek Sýs

Institute of Mathematics and Applied Informatics,
Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology in Bratislava,
Ilkovičova 3, 812 19 Bratislava, Slovak Republic
{michal.mikus,marek.sys}@stuba.sk

Abstract. In this paper we examine the first working implementation of a fully homomorphic scheme from C.Gentry and S.Halevi. We implemented the ciphertext-only attack from [2] using the NTL library and show that only dimensions up to 128 are feasible for common computational power. We propose also two improvements of this attack that enable us to use the fastest variant of LLL from NTL and compare the results.

1 Introduction

The area of homomorphic cryptosystems is one of the most focused topics in cryptography. The first reason behind this is their wide application scope. However, main interest was caused by Gentry's work [6], [7], where he introduced the idea of the first working fully homomorphic cryptosystem. Subsequent works in this area were either instantiantions of his idea [19], [5], [1], [8], improvements [20],[4] or cryptanalysis [11], [16], [2].

Our Contribution. The goal of this paper is to examine the somewhat homomorphic scheme from [8] and implement the attack introduced in [2]. The cryptanalysis in [2] concludes that the Gentry's cryptosystem is not secure for dimensions up to 8192, but we show that the proposed attack is not practical for dimensions higher than 256. We also propose two improvements of the attack that significantly reduce the running time for dimensions up to 128.

Related Work. A similar attack on Gentry's scheme using the lattice reduction algorithms was described in [16]. Its goal, however, was to recover the secret key and the attack used only standard LLL and BKZ algorithms for the reduction. The attack was also successful for dimensions up to 128.

* This work was supported by the grant APVV-0586-11.

2 Preliminaries

In this paper we consider vectors b_1, b_2, b_3 as rows and matrices as columns of the corresponding vectors $B = (b_1, b_2, b_3)^T$. By $\|b\|$ we denote the usual Euclidean norm of a vector b.

For any $a, b \in \mathbf{Z}$, $[a]_b$ denotes the reduction modulo b mapped into the interval $[-b/2, b/2)$. For example 8 mod 3 = 2, but $[8]_3 = -1$. For any real number $x \in \mathbf{R}$, the rounding of x to the nearest integer is denoted by $\lfloor x \rceil$. The number 1/2 is rounded up. These notations are naturally generalized to vectors and matrices: for a vector v the expression $\lfloor v \rceil$ denotes the rounding operation applied to every coefficient of v.

2.1 Homomorphic Cryptosystems

A cryptosystem is called homomorphic if it provides a way how to perform some computations on the plaintexts using only the corresponding ciphertexts. This computations can involve addition or multiplication or both. Formally, homomorphic cryptosystem was defined in [14] as follows:

Definition 1. *[14] Let the message space $(M, \circ)$ be a finite (semi-)group and let σ be the security parameter. A homomorphic public-key encryption scheme (or homomorphic cryptosystem) on M is a quadruple (K, E, D, A) of probabilistic, expected polynomial time algorithms, satisfying:*

Key generation: *On input[1] 1^σ algorithm K outputs an encryption/decryption key pair $k = (k_e, k_d) \in \mathcal{K}$ where $\mathcal{K}$ denotes the key space.*

Encryption: *On inputs $1^\sigma, k_e$ and $m \in M$ the encryption algorithm E outputs a ciphertext $c \in \mathcal{C}$ where $\mathcal{C}$ denotes the ciphertext space.*

Decryption: *The decryption algorithm D is deterministic. On inputs $1^\sigma, k$ and $c \in C$ it outputs $m \in M$ so that:*

$$\forall m \in M : \text{ if } c = E(1^\sigma, k_e, m) \text{ then } P[D(1^\sigma, k, c) \neq m] \text{ is negligible.}$$

Homomorphic property: *A is an algorithm that on inputs $1^\sigma, k_e$ and elements $c_1, c_2 \in \mathcal{C}$ outputs an element $c_3 \in \mathcal{C}$ so that*

$$\forall m_1, m_2 \in M\text{: if } m_3 = m_1 \circ m_2 \text{ and } c_1 = E(1^\sigma, k_e, m_1) \text{ and } c_2 = E(1^\sigma, k_e, m_2) \\ \text{then } P[D(1^\sigma, k, c_3) \neq m_3] \text{ is negligible.}$$

The operation $\circ$ defined in the plaintext space M is addition or multiplication in practice. Then the homomorphic cryptosystem is called additively, resp. multiplicatively homomorphic and we denote the homomorphic algorithms A_+ and $A_\times$ respectively.

[1] The notation 1^σ has pure formal purpose, since the running time of algorithm is derived from the length of the input.

Definition 2. *A tuple* $(K, E, D, A_+, A_\times)$ *of probabilistic, expected polynomial time algorithms is called fully (or algebraically) homomorphic cryptosystem,* (K, E, D, A_+) *is additively homomorphic and quadruple* $(K, E, D, A_\times)$ *is multiplicatively homomorphic cryptosystem.*

The first proposal of a fully homomorphic cryptosystem was published by C.Gentry in [6]. Gentry's new idea was to construct first a "somewhat homomorphic cryptosystem" (SHS), that can support both additions and multiplications, but only a limited number of them. Gemtry further showed how to use the SHS to construct a fully homomorphic scheme (FHS), that can support unlimited number of additions and multiplications. This construction of FHS from SHS is called bootstrapping.

The basic idea of such a somewhat homomorphic scheme was described in [6]. The scheme was based on ideal lattices[2] with the implementation details omitted. In [8] the first working implementation of SHS was described.

We describe the basic idea of the scheme first. The somewhat homomorphic scheme is a lattice based cryptosystem described in [9]. The underlying structure is a lattice $\mathcal{L}$ over integers, the secret key is a "near orthogonal" basis B_s of $\mathcal{L}$ and the public key is the Hermite normal form of the secret basis $B_p = HNF(B_s)$. The encryption procedure encodes the message into a small error vector, picks a random vector of the lattice and outputs the sum of these two vectors as the ciphertext. The decryption algorithm then finds the closest lattice vector to the ciphertext, computes the error vector and outputs the message. The success of the decryption procedure is based on the orthogonality of the secret basis and on the fact, that the error vector is small enough. To be precise we define r_{dec} as the radius of the largest n-dimensional ball that can be fitted to the parallelepiped $\mathcal{P}(B_s)$ (n is the dimension of the lattice). When the error vector is smaller than r_{dec}, then the ciphertext can be simply mapped to the parallelepiped $\mathcal{P}(B_s)$ and its coefficients rounded to zero.

It can be easily seen that sum of two ciphertexts c_1, c_2 produces again a ciphertext c_3, whose error vector e_3 is the sum of the two corresponding error vectors. When the new error e_3 is still smaller than r_{dec}, then the decryption will output the message m_3 corresponding to e_3. The multiplication of two lattice points is not defined, so the authors extended this lattice based scheme to the ideal lattices and defined multiplication naturally in the corresponding ring.

In the following we describe the proposed SHS. The plaintext-space of the scheme is $\mathcal{P} = \{0, 1\}$, the parameters of the SHS are integers N, t, where N is some power of two and t represents the bitlength of the underlying polynomial $v(x)$. The secret and public keys should be two bases of the same N-dimensional lattice, but authors showed how to effectively represent these bases. The public key consists of two numbers d and r and the secret key is a single number w_i. The bitlength of these numbers is approx $N \cdot t$. The ciphertext-space is the ring $\mathbf{Z}_d$.

[2] The notion "ideal lattice" means simply some ideal I of some ring $\mathbf{Z}[x]/(f(x))$, for suitable polynomial $f(x)$ of degree n. As every member of the ideal is a fixed-degree polynomial, it can be represented by its vector of coefficients. In this way it represents a point in the vector space $\mathbf{Z}^n$ and all members of the ideal I represent some discrete lattice.

Authors recommend a small, medium and large parameter settings (with $N = 2048$, $N = 8192$ and $N = 32768$ respectively and a common choice $t = 380$), but in this paper we focus only on the toy versions of the system with $N \in \{64, 128, 256\}$ and $t \in \{64, 96, 128\}$. The basic algorithms of the scheme are:

Keygen(N, t)

1. set $f(x) = x^N + 1$
2. choose a random polynomial $v(x)$ of degree $(N-1)$, with a t-bit coefficients, s.t. $v(x)$ and $f(x)$ have a single root r in common
3. compute $w(x)$ s.t. $w(x)v(x) \equiv d \bmod f(x)$, where $d = \mathit{resultant}(f(x), v(x))$
4. output $PK = (N, t, d, r)$ and $SK = (w_{i_0})$, where $w_{i_0} \equiv 1 \bmod 2$ is some coefficient of $w(x)$

Authors also provided an effective algorithm that implemented the computation of $w(x)$ in the third step and a way how to check the common root of the random $v(x)$ and $f(x)$. We omit the technical details here.

Encrypt(PK, m, q)

1. choose random $u(x) \in \mathbf{Z}[x]$ of degree $(N-1)$ where $u_i = \pm 1$ with probability $(1-q)$ and $u_i = 0$ with probability q
2. set $c(x) = m + 2 \cdot u(x)$
3. output $c = [c(r)]_d$

The input parameter $q \in (0, 1)$ of the encryption procedure controls the mapping of the message $m \in \mathcal{P}$ to the error polynomial $u(x)$. The recommended value of this parameter is $q = 1 - 20/N$, so that approximately 20 coefficients will be ± 1 and the rest zero. This ensures that the initial error vector is small and the scheme can support a higher number of additions and multiplications.

Decrypt(SK, c)

1. $m = [c \cdot w_{i_0}]_d$
2. output $m \bmod 2$

This simple computation of the first step represents the mapping of the ciphertext vector $(c, 0, \ldots, 0)$ to the initial parallelepiped $\mathcal{P}(B_s)$ and the second step extraction of the message from the error vector. The detailed proof can be seen in [8].

Add(c_1, c_2, d)

1. $c = c_1 + c_2$
2. output $[c]_d$

Multiply(c_1, c_2, d)

1. $c = c_1 \cdot c_2$
2. output $[c]_d$

3 Ciphertext-Only Attack on SHS

In this section we describe our ciphertext-only attack on Gentry's somewhat homomorphic scheme. The somewhat homomorphic scheme is based on discrete lattices [13]. We explain the necessary notions also in appendix A.

3.1 Gu's Attack

The attack described in [2] is based on the observation that the encryption algorithm can be rewritten as $c = m + 2\sum u_i r^i + k \cdot d$ for some integer k. The attack is a standard search for an integer combination using a lattice reduction algorithm. The plaintext m, coefficients $u_i \in \{-1, 0, 1\}$ and k are the only unknowns in the encryption equation. Since the public key contains r, the attacker knows r^i for every i and needs to find "only" an integer combination[3] of c, d, $2 \cdot r^i$ that sums to 0 or 1 (i.e., the plaintext).

The attack from [2] used the following $(N+1) \times (N+2)$ matrix:

$$\begin{pmatrix} c & 1\,0\,0\ldots0 \\ \alpha_1 & 0\,1\,0\ldots0 \\ \alpha_2 & 0\,0\,1\ldots0 \\ \vdots & 0 \quad\ \ddots\ 0 \\ \alpha_{N-1} & 0\,0\,0\ldots0 \\ d & 0\,0\,0\ldots1 \end{pmatrix},$$

where $\alpha_i = [2 \cdot r^i]_d$.

The rows of the matrix form a basis of some lattice L. It can be shown that vector $a = (m, 1, u_1, \ldots, u_{N-1}, k)$ belongs to the lattice. Further $m \in \{0, 1\}$, $u_i \in \{-1, 0, 1\}$ (where approximately 20 coefficients u_i are nonzero) and $k \in \mathbf{N}$ is symmetrically distributed around 0 with maximal value of 20. The vector a is therefore very short when compared to the initial vectors of the lattice. Moreover, its length is much smaller than the theoretical bound that is proved for both LLL and BKZ algorithms. Therefore, there is a non-zero probability that the LLL algorithm finds this vector and assuming that a is actually the shortest vector of the lattice, the probability is known to be quite high for small dimensions.

The paper [2] omitted any practical results and simply concluded that the Gentry's system is insecure up to $N = 8192$.

3.2 Our Approach

For our attack, we use an $(N+1) \times (N+1)$ matrix:

[3] The situation is essentially the same as in the knapsack cryptosystem.

$$B = \begin{pmatrix} c & 0\,0\ldots 0 \\ \alpha_1 & 1\,0\ldots 0 \\ \alpha_2 & 0\,1\ldots 0 \\ \vdots & \ddots\, 0 \\ \alpha_{N-1} & 0\,0\ldots 0 \\ d & 0\,0\ldots 1 \end{pmatrix},$$

where $\alpha_i = [2{\cdot}r^i]_d$. It can be seen that B can be obtained from the one used by Gu by omission of the second column. As the special vector $a = (m, u_1, \ldots, u_{N-1}, k)$ is in the lattice generated by the matrix, this modification does not affect the performance of the attack.

The idea of the attack is the same as in [2], to find the vector a by employing reduction algorithms. The success of any reduction algorithm depends on how many short vectors other than a are within the proved bounds of the algorithm ($1.02^n \cdot det(L)^{1/n}$ for the LLL algorithm). The number of such short vectors is generally unknown, so we tested various algorithms and their settings.

Our experiments are based on the widely established NTL library [15] which offers optimized versions of both BKZ and LLL algorithms. As the LLL algorithm is faster[4] than BKZ and we still expect the LLL algorithm to find the vector a we will use only the LLL algorithm in our experiments.

Experiment setting. We generated nine instances of the cryptosystem for three "toy" dimensions 64, 128 and 256 with $t \in \{64, 96, 128\}$. For each instance of the cryptosystem we generated several PT-CT pairs along with the private encryption data $u(x)$. The coefficients of the polynomial $u(x)$ will be used for a correctness check of the reduced vector b_1. The attack scenario is following:

– generate matrix B from public parameters and ciphertext (the first column entries have 4000 – 33000 bitlength)
– reduce B with a reduction algorithm
– check the correctness of the first vector of the reduced matrix

It should be noted that the bitlength of numbers in the first column is proportional to $N \cdot t$ and that the determinant of the lattice is equal to the ciphertext $c \approx 2^{N \cdot t}$.

We say that the attack is successful when the first vector of the reduced matrix b_1 contains all the coefficients of $u(x)$ and that the attack fails otherwise. We check also the case when the vector b_1 contains all negated coefficients of $u(x)$ (multiplied by -1). We didn't base the evaluation of the success rate on the computed plaintexts, because the plaintext space is $\{0, 1\}$ and thus the correct plaintext is still obtained in approximately half of the failed attempts.

Standard LLL Attack. At first we tested the standard LLL algorithm for the reduction. As expected, the LLL was successful but slow: the dimension $N = 64$

[4] The BKZ is expected to output bases of better quality, i.e. shorter and more orthogonal vectors.

and $t = 64$ had success rate 16/16 and average running time 5200 seconds ≈ 1.44 hours on a standard desktop computer (2.6GHz Intel processor, 4GB RAM). A single attack in the dimension $N = 128$ and $t = 64$ took about 160 000 seconds ≈ 44 hours (with success). The expected running time for dimension 256 is about 55 days on our desktop computer. For a comparison this attack on dimension 256 was also successfully performed by Gu with running time of 22 days in their supercomputing center [3].

Floating Point Versions. To reduce the running time of the LLL algorithm we tested the floating point variants of LLL from the NTL library, namely `LLL_XD` and `LLL_FP` versions. These two variants differ only in the used precision. The `LLL_FP` is faster, but can cause some round-off errors and overflows and accepts only numbers up to 500 bits as an input. The `LLL_XD` version works for large numbers (more than 20000 bitlength) and was chosen as a trade-off between the precise LLL algorithm and the fast `LLL_FP` variant.

1. `LLL_FP_cut`: because the bitlength of the initial matrix coefficients was too long even for our small experiments, we further simplified the FP-attack. The first idea is to shorten each vector of the initial matrix by the same factor and let the algorithm find the shortest vector in the modified lattice. If the shortening could be done with exact precision, we would of course obtain the same shortest vector in the modified lattice (divided by the same factor as every vector in the basis). However, this would not shorten the bitlength of the numbers in the matrix. Therefore, we cut-off the $(l_{max} - 500)$ *least* significant digits of each number in the first column, where l_{max} is the maximal bitlength in the matrix and we hope that the error introduced by the cutting would not be too large. We denote this variant as `LLL_FP_cut` attack and we expect it to have the lowest success rate.
2. `LLL_FP_block`: the second idea is to split the large numbers in the first column into several blocks of 500 bits and modify the lattice so that the shortest vector in the modified lattice corresponds to the shortest vector of the original one. We believe that this idea is new to the area of lattice reduction and our results will inspire further ideas and more effective algorithms. We illustrate this idea of splitting large numbers on the following example.

   Example 1. For simplicity assume, that the bitlength of the matrix entries varies between 1200 and 1400, so we split the numbers into 3 blocks. We obtain this $(N + 3) \times (N + 3)$ matrix:

$$\begin{pmatrix} c_1 & c_2 & c_3 & 0\,0\ldots0 \\ \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & 1\,0\ldots0 \\ \alpha_{2,1} & \alpha_{1,2} & \alpha_{2,3} & 0\,1\ldots0 \\ \vdots & \vdots & \vdots & \ddots\,0 \\ \alpha_{N-1,1} & \alpha_{N-1,2} & \alpha_{N-1,3} & 0\,0\ldots0 \\ d_1 & d_2 & d_3 & 0\,0\ldots1 \\ -1 & 2^{500} & 0 & 0\,0\ldots0 \\ 0 & -1 & 2^{500} & 0\,0\ldots0 \end{pmatrix}.$$

The last two vectors in the matrix can be used to compensate the carry between blocks i and $i+1$. Let c_1 be the most significant block of the number c. It can be shown that the linear combination of vectors selected by the coefficients of $u(x)$ results into a vector $a' = (0, 0, m, u_1, u_2, \ldots, u_{N-1}, k)$ in this modified matrix and therefore this vector can be found by the `LLL_FP` algorithm.

We call this modification a `LLL_FP_block` variant and we expect it to have lower running time than the `LLL_XD` algorithm.

3.3 Results

In this section we display the obtained results. We categorize the results by the dimension, as it has the biggest impact on the running time. We begin with the dimension $N = 64$:

dimension	bitlength t	reduction method	avg.time(s)	success rate
64	64	LLL_XD	49	30/30
64	96	LLL_XD	100	30/30
64	128	LLL_XD	172	30/30
64	64	LLL_FP_block	24	16/30
64	96	LLL_FP_block	50	16/30
64	128	LLL_FP_block	88	17/30
64	64	LLL_FP_cut	0.5	30/30
64	96	LLL_FP_cut	0.5	30/30
64	128	LLL_FP_cut	0.5	30/30

We can see that the methods fulfilled the expectations in the running times. The `LLL_XD` is 100× faster than the precise LLL algorithm and the `LLL_FP_cut` variant is even 100× faster than the XD algorithm. The `LLL_FP_block` variant is only two times faster than `LLL_XD`, this may stem from the slightly increased dimension of the matrix.

The success rate was determined from 30 attempts with different ciphertexts. Both `LLL_XD` and `LLL_FP_cut` have a 100% success rate. This is surprising in the case of `LLL_FP_cut` method, we conclude that the computation of the LLL algorithm hadn't too many steps and the error didn't grow too big during the execution of the algorithm. We note that the time for `LLL_FP_cut` method is constant for various t, because the matrix entries are cut to a constant length of 500.

The `LLL_FP_block` method has a success rate only little over 50% and further examination of the obtained results showed that in the cases of failure a completely different vector were found. These vectors were small enough to satisfy the theoretical expectation, but bigger than the desired vector a. We speculate that this is the case when the reduction algorithm doesn't output the shortest vector.

Next we display the results for dimension $N = 128$, the success rate was determined from 20 (resp. 12 for the largest setting) attempts:

dimension	bitlength t	reduction method	avg.time(s)	success rate
128	64	LLL_XD	1580	20/20
128	96	LLL_XD	3800	20/20
128	128	LLL_XD	7800	12/12
128	64	LLL_FP_block	515	9/20
128	96	LLL_FP_block	1090	9/20
128	128	LLL_FP_block	1980	10/20
128	64	LLL_FP_cut	4.2	3/20
128	96	LLL_FP_cut	4.2	3/20
128	128	LLL_FP_cut	4.2	3/20

The results show the expected increase in the average time for the `LLL_XD` method. The success rate is still 100%.

More interesting are the results for the `LLL_FP_block` variant. The running time compared to the XD variant is three times shorter, which is improvement from the case $N = 64$. The success rate is still about 50%.

The `LLL_FP_cut` method has the lowest running time, only 8 times slower than in the dimension $N = 64$ and still has a non-zero probability of finding a correct solution. This makes it the best candidate for a parallel attack in the dimension $N = 128$: we can generate K permutations of the original basis vectors, try to solve each problem separately and see, if we get a solution. It is known that the LLL algorithm (and its variants) behaves differently on a permutation of the input vectors. Under the assumption that each run of the LLL algorithm on a permutation of the basis vectors is independent from the previous, the success rate is expected to be $1 - (17/20)^K$ for dimension $N = 128$, with an overall running time of 4 seconds.

We don't display the results for dimension $N = 256$, because both the floating point versions `LLL_XD` and `LLL_FP_block` failed to reduce even the lowest setting (with $t = 64$). The error messages indicate that a too low precision was selected and the algorithm ended in an infinite loop. We conclude that for higher dimensions a slower and more precise algorithm has to be created. The attack with `LLL_FP_cut` method finished in 30 seconds, but the success rate was zero.

4 Conclusions and Open Questions

We presented practical results of a ciphertext-only attack on a somewhat homomorphic scheme of Gentry-Halevi. The attack was inspired by the paper [2]. Since the average running times were obtained from a standard desktop computer, we use them only to compare the effectiveness of the proposed variants. We showed that the two modifications of the general attack(`LLL_FP_cut` and `LLL_FP_block`) considerably improved the attack statistics for the toy dimensions 64 and 128.

The modifications that reduce the running time at the expense of a lower success rate can be further combined with parallelization techniques. These ideas can be also applied to another variant of the LLL algorithm and we believe that this kind of attack can be successful also for higher dimensions.

A Discrete Lattices

Definition 3. *Let $n, m \in \mathbf{N}$, $n \leq m$, further let $b_1, \ldots, b_n \in \mathbf{R}^m$ be n linearly independent vectors and $B = (b_1, \ldots, b_n)^T$ a corresponding matrix. Then lattice $\mathcal{L}$ spanned by these vectors is defined as*

$$\mathcal{L} = L(B) = \left\{ \sum_{i=1}^{n} x_i b_i : x_i \in \mathbf{Z} \right\},$$

and we say that the lattice $\mathcal{L}$ has dimension n and basis B.

A lattice can have more that one basis. For any two bases B_1 and B_2 of the same lattice holds that $B_1 = U_1 \cdot B_2$ and $B_2 = U_2 \cdot B_1$ for some integer matrices U_1, U_2. From these equations follows that $det(B_1) = det(U_1) \cdot det(B_2) = det(U_1) \cdot det(U_2) \cdot det(B_1)$ and since $det(U_i) \in \mathbf{Z}$, then $det(U_i) = \pm 1$ and therefore $|det(B_1)| = |det(B_2)|$. The determinant of lattice $L(B)$ is defined as $\sqrt{det(BB^T)}$. If for a lattice $\mathcal{L}$ $n = m$ then we say $\mathcal{L}$ is full-rank lattice and $det(\mathcal{L}) = |det(B)|$ for any basis B of $\mathcal{L}$.

Definition 4. *Let $\mathcal{L}$ be a lattice and B some basis. Then a basic parallelepiped of the basis B is defined as*

$$\mathcal{P}(B) = \left\{ \sum_{i=1}^{n} x_i b_i : x_i \in \langle -1/2, 1/2) \right\}.$$

The volume of the basic parallelepiped corresponds to the determinant of the lattice. Although the volume of $\mathcal{P}(B)$ is invariant for any basis, the geometrical shape of $\mathcal{P}(B)$ depends on the basis.

For any full-rank lattice $\mathcal{L}$ holds that the lattice tiles the vector space $\mathbf{R}^n$ into parallelepipeds of the same shape that are only transitions of the basic parallelepiped by some lattice vector. This tiling induces mapping that maps arbitrary $x \in \mathbf{R}^n$ into the basic parallelepiped of $\mathcal{L}$ by addition of some lattice vector. For any point $x \in \mathbf{R}^n$ and for any full-rank lattice $\mathcal{L}$ with a basis B it is easy to compute the image of x in the basic parallelepiped $\mathcal{P}(B)$, $x' = x - B \cdot \lfloor B^{-1} \cdot x \rceil$. This x' is unique and it is usually denoted by $x \bmod B$.

For a lattice $\mathcal{L}$, the most interesting vector from cryptanalytic point of view is the shortest nonzero vector of $\mathcal{L}$[5], denoted as $\lambda_1(\mathcal{L})$.

The problem of finding the shortest vector (SVP) of an arbitrary lattice is known[6] to be NP-hard [12].

There are two approximation versions of the SVP where the goal is to find a vector $v \in \mathcal{L}$ such that:

[5] Because it usually represents a solution to some cryptosystem, e.g. secret key or plaintext message.

[6] It was first conjectured to be NP-hard in 1981 by van Emde Boas, but remained open problem for 20 years.

- $\|v\| \leq \gamma\lambda_1(\mathcal{L})$, where γ is some approximation factor. This problem is denoted as γ-SVP
- $\|v\| \leq \gamma det(\mathcal{L})^{1/n}$, γ is again some approximation factor. This problem is called an Hermite-SVP and it is commonly used when the length of the shortest vector is not known.

Both these approximation problems were successfully solved in [10] with the famous LLL algorithm. The LLL algorithm returns a reduced basis of a lattice, where $\|b_i\| \leq \|b_{i+1}\|$ for every i. The LLL algorithm is also used for finding the shortest vector and for it was showed that $\|b_1\| \leq 1.02^n \cdot det(L)^{1/n}$, so the LLL algorithm solves the Hermite-SVP with approximation factors that are exponential in the lattice dimension. The running time of the LLL algorithm is $O(m^5 n \log^3(B^*))$, where n is the dimension of the lattice and m is the dimension of each vector and B^* is the maximum of the Euclidean norm of the input vectors b_i.

Later [18] was developed the Block-Korkin-Zolotarev (BKZ) algorithm that is a blockwise variant of the LLL algorithm and is currently known as best algorithm for lattice reduction. The running time of BKZ is greater than of LLL, but it generally outputs better basis, i.e. shorter and more orthogonal basis vectors. BKZ has an additional parameter β – the blocksize and it is shown [17] that $\|b_1\| \leq (\gamma_\beta)^{\frac{n-1}{\beta-1}}$, where γ_β is the Hermite constant for dimension β. Higher blocksize β leads to shorter basis vectors, but it also increases the running time of the algorithm.

References

1. Gu, C.: New Fully Homomorphic Encryption over the Integers. Cryptology ePrint Archive, Report 2011/118, (September 21, 2011), http://eprint.iacr.org/2011/118
2. Gu, C.: Cryptanalysis of the Smart-Vercauteren and Gentry-Halevis Fully Homomorphic Encryption. IACR Cryptology ePrint Archive 2011: 328 (2011)
3. Gu, C.: Personal Communication (2012)
4. Coron, J.S., Naccache, D., Tibouchi, M.: Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers. Cryptology ePrint Archive, Report 2011/440 (2011), http://eprint.iacr.org/2011/440 (July 29, 2012)
5. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
6. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC 2009), Bethesda, USA, pp. 169–178 (2009)
7. Gentry, C.: A fully homomorphic encryption scheme. Dissertation Thesis, Stanford University (September 2009)
8. Gentry, C., Halevi, S.: Implementing Gentry's Fully-Homomorphic Encryption Scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011)

9. Goldreich, O., Goldwasser, S., Halevi, S.: Public-Key Cryptosystems from Lattice Reduction Problems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 112–131. Springer, Heidelberg (1997)
10. Lenstra, A., Lenstra, H., Lovasz, L.: Factoring polynomials with rational coefficients. Mathematische Annalen 4, 515–534 (1982)
11. Loftus, C., May, A., Smart, N.P., Vercauteren, F.: On CCA-Secure Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2010/560 (2010), http://eprint.iacr.org/2010/560 (September 21, 2011)
12. Micciancio, D.: The shortest vector problem is NP-hard to approximate to within some constant. SIAM Journal on Computing 30(6), 2008–2035 (2001)
13. Nguyen, P.Q., Valée, B.: LLL algorithm, Survey and Applications. Springer (2010)
14. Rappe, D.K.: Homomorphic cryptosystems and their applications. PhD Thesis, University of Dortmund, Dortmund, Germany (2004)
15. Shoup, V.: A library for doing Number Theory, v.5.5.2. New York University, New York (July, 29, 2012), http://shoup.net/ntl/
16. Schmidt, P.: Fully Homomorphic Encryption - Overview and Cryptanalysis. Diploma Thesis, University of Dortmund, Dortmund, Germany (2011)
17. Schnorr, C.P.: Block reduced lattice bases and successive minima. Combinatorics, Probability & Computing 3, 507–552 (1994)
18. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Mathematical Programming 66, 181–199 (1994)
19. Smart, N.P.,Vercauteren, F.: Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. Cryptology ePrint Archive, Report 2009/571 (2009), http://eprint.iacr.org/2009/571 (September 21, 2011)
20. Stehlé, D., Steinfeld, R.: Faster Fully Homomorphic Encryption. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 377–394. Springer, Heidelberg (2010)

Grover's Algorithm with Errors

Andris Ambainis, Artūrs Bačkurs, Nikolajs Nahimovs, and Alexander Rivosh*

Faculty of Computing, University of Latvia, Raina bulv. 19, Riga, LV-1586, Latvia

Abstract. Grover's algorithm is a quantum search algorithm solving the unstructured search problem of size n in $O(\sqrt{n})$ queries, while any classical algorithm needs $O(n)$ queries [3].

However, if query has some small probability of failing (reporting that none of the elements are marked), then quantum speed-up disappears: no quantum algorithm can be faster than a classical exhaustive search by more than a constant factor [8].

We study the behaviour of Grover's algorithm in the model there query may report *some* marked elements as unmarked (each marked element has its own error probability, independent of other marked elements).

We analyse the limiting behaviour of Grover's algorithm for a large number of steps and prove the existence of limiting state ρ_{lim}. Interestingly, the limiting state is independent of error probabilities of individual marked elements. If we measure ρ_{lim}, the probability of getting one of the marked states $i_1, \ldots, i_k$ is $\frac{k}{k+1}$. We show that convergence time is $O(n)$.

1 Introduction

Grover's algorithm is a quantum search algorithm solving the unstructured search problem. The algorithm works in the following model. We have an unstructured search space of n elements in which some elements have a certain property. We call these elements *marked*. We are given a procedure (*an oracle*) for checking whether an element is marked. This procedure is given as a black box that answers queries. It receives i and answers whether the i^{th} element is marked. In the quantum case, the algorithm is allowed to input superposition consisting of multiple i.

Grover's algorithm solves the unstructured search problem in $O(\sqrt{n})$ queries. It is known that any deterministic or randomized algorithm needs linear time (number of queries) to solve the above problem. Thus, Grover's algorithm provides a significant speed-up over any classical algorithm.

There has been a number of papers studying Grover's algorithm in the presence of errors of various forms. Regev and Schiff have shown [8] that if query has some small probability of failing (reporting that none of the elements are

* AA, NN and AR are supported by the European Social Fund within the project 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044. AB is supported by FP7 FET-Open project QCS.

marked), then quantum speed-up disappears: no quantum algorithm can be faster than a classical exhaustive search by more than a constant factor.

In this paper we study the behaviour of Grover's algorithm in the model there query may report *some* marked elements as unmarked. In our case each marked element has its own probability of failing, independent of other marked elements. We assume that faults are one-sided. That is, if the i^{th} element is not marked, the black box always answers that it is not marked. If the i^{th} element is marked, the black box may give the correct answer (with probability $1-p_i$) or mistakenly answer that the element is not marked (with probability p_i).

Given the importance of Grover's algorithm, we think that it is interesting to find out what exactly happens if we run Grover's algorithm in this model.

Let k be the number of marked elements. We show that if Grover's algorithm is run for a large number of steps, then the state of the algorithm converges to a mixed state that is a mixture of $|i\rangle$ for each marked i with probability $\frac{1}{k+1}$ each and the uniform superposition of all non-marked elements with probability $\frac{1}{k+1}$. Surprisingly, the final state is independent of the error probabilities of different marked elements. Initially, the probabilities of finding the elements with higher probabilities of correct answer grow faster but, in the limit for a large number of steps the probabilities of finding all elements i converge to the same value $\frac{1}{k+1}$.

We also quantify the speed of convergence: it happens in $O(n)$ steps. This matches the lower bound of [8][1].

Related Work. The work of Regev and Schiff [8] mentioned above is the paper that is most closely related to our work.

Several authors [5,9,10] have studied the effect of random imperfections in either diffusion transformation or black box query on the performance of Grover's algorithm, showing that such type of noise can completely destroy the advantage of Grover's algorithm over classical exhaustive search. The difference between their work and our work is that they consider small random imperfections that occur in every step of the algorithm while we consider the case there query is performed correctly for some marked elements and not performed at all for others.

Buhrman et al. [2] have looked at a *coherent noise* model in which the algorithm has access to a set of unitary procedures A_i that check whether the i^{th} element is marked and have some probability of error. The algorithm is allowed to run both A_i and A_i^{-1} multiple times. This model is sufficiently general to enable a fault-tolerant computation and allows to simulate any noise-free quantum algorithm that makes T queries by a noisy algorithm that makes $O(T \log T)$ queries. In some cases, a constant overhead instead of a logarithmic one is sufficient. The difference between coherent noise and our models is that in coherent noise model the state after query is still a pure state, while in our model query leads to a mixes state.

[1] Technically, the lower bound of [8] is for a slightly different model but the difference between the models is not important in this case.

2 Technical Preliminaries

We use the standard notions of quantum states, density matrices, etc., as described in [6] or [7].

Grover's algorithm[3]

Suppose we have an unstructured search space of size n. Grover's algorithm starts with a starting state $|\psi_{start}\rangle = \frac{1}{\sqrt{n}}\sum_{i=1}^{n}|i\rangle$. Each step of the algorithm consists of two transformations: Q and D. Here, Q is a query to a black box defined by

– $Q|i\rangle = -|i\rangle$ if i is a marked element;
– $Q|i\rangle = |i\rangle$ if i is not a marked element.

D is the diffusion transformation described by the following $n \times n$ matrix:

$$D = \begin{pmatrix} -1+\frac{2}{n} & \frac{2}{n} & \dots & \frac{2}{n} \\ \frac{2}{n} & -1+\frac{2}{n} & \dots & \frac{2}{n} \\ \dots & \dots & \dots & \dots \\ \frac{2}{n} & \frac{2}{n} & \dots & -1+\frac{2}{n} \end{pmatrix}.$$

We refer to $|\psi_t\rangle = (DQ)^t|\psi_{start}\rangle$ as the state of Grover's algorithm after t time steps.

Grover's algorithm has been analysed in detail and many facts about it are known [1]. If there is one marked element i, the probability of finding it by measuring $|\psi_t\rangle$ reaches $1-o(1)$ for $t = O(\sqrt{n})$. If there are k marked elements, the probability of finding one of them by measuring $|\psi_t\rangle$ reaches $1-o(1)$ for $t = O(\sqrt{n/k})$.

Frobenius norm[4]

Let $\rho = (\rho_{ij})$ be an $n \times n$ matrix. The *Frobenius norm* (also called *Euclidean norm* or *l_2-norm*) of ρ is defined as

$$\|\rho\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}|\rho_{ij}|^2}.$$

Frobenius norm is unitarily invariant: if U unitary, then $\|U\rho\|_F = \|\rho\|_F = \|\rho U\|_F$ [4, chapter 5.6]. Also, $\|\rho\|_F \geq 0$ and $\|\rho_1+\rho_2\| \leq \|\rho_1\| + \|\rho_2\|$, as for any matrix or vector norm.

3 Grover's Algorithm with Errors

We assume that a search space of size n contains k marked elements $i_1, i_2, \dots, i_k$. In each step, instead of the correct query Q, we apply a faulty query (*faulty oracle*) Q' defined as follows:

– $Q'|i_j\rangle = |i_j\rangle$ with probability p_j;
– $Q'|i_j\rangle = -|i_j\rangle$ with probability $1-p_j$;
– $Q'|i\rangle = |i\rangle$ if i is not a marked element.

For different elements i_j, faults occur independently one from another. Also, for different steps faults are independent.

We show

Theorem 1. *Let ρ_t be the density matrix of state of Grover's algorithm with a faulty oracle after t queries. Then, the sequence $\rho_1, \rho_2, \ldots$ converges to*

$$\rho_{lim} = \frac{1}{k+1}\sum_{j=1}^{k} |i_j\rangle\langle i_j| + \frac{1}{k+1}|\phi\rangle\langle\phi|$$

where $|\phi\rangle = \frac{1}{\sqrt{n-k}}\sum_{i\neq i_j}|i\rangle$ is the uniform superposition over all non-marked i.

If we measure ρ_{lim}, the probability of getting one of the marked states $i_1, \ldots, i_k$ is $\frac{k}{k+1}$. Interestingly, the final state is independent of the error probabilities $p_1, \ldots, p_k$. Initially the probabilities of finding the elements with higher probabilities of correct answer grow faster but, in the limit for a large number of steps, the probabilities of finding all elements i_j converge to the same value $\frac{1}{k+1}$.

The next result quantifies the speed of convergence to the limiting state ρ_{lim}.

Theorem 2. *Assume that errors occur with the same probability $p_1 = \ldots = p_k = p$ for all marked elements. Then, for every $\epsilon > 0$, there exists $t = O(n)$ such that if we run Grover's algorithm with a faulty oracle for t steps and measure the result, we get one of the marked elements with probability in $[\frac{k}{k+1} - \epsilon, \frac{k}{k+1} + \epsilon]$.*

4 Limiting Behaviour of Grover's Algorithm with Errors

In this section we will study limiting behaviour of Grover's algorithm with errors for large number of steps and will prove the Theorem 1.

Consider the density matrix ρ_t of the quantum state of Grover's algorithm after t queries. Due to symmetry, we can assume that the first k basis states correspond to the marked elements. Note that Grover's algorithm acts in the same way on all unmarked elements. Therefore, the state of the algorithm is a probabilistic mixture of pure states of the form

$$\alpha_1|1\rangle + \ldots + \alpha_k|k\rangle + \sum_{i=k+1}^{n} \beta|i\rangle, \tag{1}$$

with the amplitudes of all unmarked states being equal. The density matrix ρ_t, then, takes the form

$$\rho_t = \begin{bmatrix} a_1 & b_{1,2} & b_{1,3} & \ldots & c_1 & \ldots & c_1 \\ b_{1,2} & a_2 & b_{2,3} & \ldots & \vdots & \ddots & \vdots \\ b_{1,3} & b_{2,3} & a_3 & \ldots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & c_k & \ldots & c_k \\ c_1 & \ldots & \ldots & c_k & d & \ldots & d \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_1 & \ldots & \ldots & c_k & d & \ldots & d \end{bmatrix}$$

because the density matrix for every pure state (1) in the mixture ρ_t is of this form.

Let p_i be the error probability for the i^{th} marked element. The effect of the faulty query Q' on the density matrix ρ_t is:

$$\begin{array}{ll} a_i & \mapsto a_i \\ b_{i,j} & \mapsto (2p_i-1)(2p_j-1)b_{i,j} \\ c_i & \mapsto (2p_i-1)c_i \\ d & \mapsto d \end{array} . \tag{2}$$

Let us prove $b_{i,j} \mapsto (2p_i - 1)(2p_j - 1)b_{i,j}$. Consider the corresponding entry $(Q'\rho_t Q')_{ij}$ of the density matrix, after the faulty query Q' is applied. If Q' changes the sign of either $|i\rangle$ or $|j\rangle$, the entry is equal to $-b_{ij}$. This happens with probability $p_i(1-p_j) + p_j(1-p_i)$. If Q' changes the sign of both $|i\rangle$ and $|j\rangle$ or none of them, the entry is equal to b_{ij}. This happens with probability $p_ip_j + (1-p_i)(1-p_j)$. Hence,

$$(Q'\rho_t Q')_{ij} = -b_{ij}(p_i(1-p_j) + p_j(1-p_i)) + b_{ij}(p_ip_j + (1-p_i)(1-p_j)) =$$

$$= (1-2p_i)(1-2p_j)b_{ij}.$$

Similarly, we can prove that $c_i \mapsto (2p_i - 1)c_i$, $a_i \mapsto a_i$ and $d \mapsto d$.

Consider the Frobenius norm of the density matrix. If we multiply the density matrix by the unitary diffusion matrix, its Frobenius norm does not change. Since the faulty query transformation decreases the Frobenius norm (if $0 < p_i < 1$) and the Frobenius norm takes non-negative values, the $\lim_{t\to\infty} \|\rho_t\| = C$ exists.

If $\lim_{t\to\infty} b_{i,j} \neq 0$ we obtain a contradiction, because the Frobenius norm decreases infinitely. Analogously, we can prove that $\lim_{t\to\infty} c_i = 0$.

Let us prove $\lim_{t\to\infty}(a_i - a_j) = 0$ for each $i \neq j$. Assume it is not true, i.e. there exist $i \neq j$ and $\delta > 0$ so that $|a_i - a_j| > \delta$ for infinitely many t. Consider t' so that for all $t > t'$ and all m, l inequalities $b_{m,l} < \epsilon$ and $c_m < \epsilon$ hold. After right multiplying the density matrix by the diffusion matrix

$$\rho_t D = \begin{bmatrix} a_1 & \ldots & O(\epsilon) & O(\epsilon) & \ldots & O(\epsilon) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ O(\epsilon) & \ldots & a_k & O(\epsilon) & \ddots & O(\epsilon) \\ O(\epsilon) & \ldots & O(\epsilon) & d & \ldots & d \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ O(\epsilon) & \ldots & O(\epsilon) & d & \ldots & d \end{bmatrix} \begin{bmatrix} -1+\frac{2}{n} & \frac{2}{n} & \ldots & \frac{2}{n} \\ \frac{2}{n} & -1+\frac{2}{n} & \ldots & \frac{2}{n} \\ \ldots & \ldots & \ldots & \ldots \\ \frac{2}{n} & \frac{2}{n} & \ldots & -1+\frac{2}{n} \end{bmatrix},$$

the last column contains values $\frac{2a_1}{n} + O(\epsilon)$, ..., $\frac{2a_k}{n} + O(\epsilon)$ and $\frac{d(n-2k)}{n} + O(\epsilon)$ ($n-k$ times). After left multiplying this matrix by the diffusion matrix, each of the first k elements in the last column takes the value $2v - \frac{2a_i}{n} + O(\epsilon)$, where v is the arithmetic mean of the last column of $\rho_t D$. We obtain a contradiction by choosing a sufficiently small ϵ, because at least two of these values differ by at least $\frac{2\delta}{n} + O(\epsilon)$.

For an arbitrary ϵ we can choose t' so that for every $t > t'$ the inequalities $b_{m,l} < \epsilon$, $c_m < \epsilon$ and $|a_m - a_l| < \epsilon$ hold for all m and l. Since $a_1 + \ldots + a_k + d(n-k) = 1$ (a property of the density matrix), it follows that $a_i = \frac{1-d(n-k)}{k} + O(\epsilon)$. So, the arithmetic mean of the last column of $\rho_t D$ is

$$v = \frac{2(a_1 + \ldots + a_k) + d(n-2k)(n-k)}{n^2} + O(\epsilon) =$$

$$= \frac{2 + d(n-2k-2)(n-k)}{n^2} + O(\epsilon).$$

After left and right multiplying the density matrix by the diffusion matrix, the last column's i-th value is

$$2v - \frac{2a_i}{n} + O(\epsilon) = 2v - \frac{2-2d(n-k)}{nk} + O(\epsilon) =$$

$$= \frac{4 + 2d(n-2k-2)(n-k)}{n^2} - \frac{2-2d(n-k)}{nk} + O(\epsilon) =$$

$$= \frac{2(n-2k)(d(k+1)(n-k)-1)}{kn^2} + O(\epsilon).$$

Since this sum must be $O(\epsilon)$, it follows that $d(k+1)(n-k)-1 = O(\epsilon)$, assuming $n \neq 2k$. Choosing ϵ arbitrarily small, we obtain $\lim_{t\to\infty} d = \frac{1}{(k+1)(n-k)}$ and $\lim_{t\to\infty} a_i = \frac{1}{k+1}$. □

5 Convergence Speed of Grover's Algorithm with Errors

In this section we will study how fast Grover's algorithm with errors converges to its limiting state and will prove the Theorem 2.

We describe the quantum state of Grover's algorithm after t queries by the density matrix

$$\rho_t = \begin{bmatrix} a_1 & b_{1,2} & b_{1,3} & \ldots & c_1 & \ldots & c_1 \\ b_{1,2} & a_2 & b_{2,3} & \ldots & \vdots & \ddots & \vdots \\ b_{1,3} & b_{2,3} & a_3 & \ldots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & c_k & \ldots & c_k \\ c_1 & \ldots & \ldots & c_k & d & \ldots & d \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c_1 & \ldots & \ldots & c_k & d & \ldots & d \end{bmatrix}.$$

In this section we assume that errors occur with the same probability $p_1 = \ldots = p_k = p$ for all marked elements. Thus, the density matrix takes the much simpler form

$$\rho_t = \begin{bmatrix} a & b & b & \dots & c & \dots & c \\ b & a & b & \dots & \vdots & \ddots & \vdots \\ b & b & a & \dots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & c & \dots & c \\ c & \dots & \dots & c & d & \dots & d \\ \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ c & \dots & \dots & c & d & \dots & d \end{bmatrix}.$$

In the further analysis we use the square of the Frobenius norm of the density matrix:

$$\|\rho\|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} |\rho_{ij}|^2.$$

We will also need the function

$$S(\rho) = k(k-1)b^2 + 2k(n-k)c^2, \tag{3}$$

which gives the sum of squares of all b and c elements of the density matrix.

According to (2), the faulty query transformation Q' decreases the square of the Frobenius norm of the density matrix by

$$k(k-1)b^2 + 2k(n-k)c^2 - k(k-1)(b(2p-1)^2)^2 - 2k(n-k)(c(2p-1))^2 =$$

$$= k(k-1)b^2(1-(2p-1)^4) + 2k(n-k)c^2(1-(2p-1)^2) >$$

$$> (k(k-1)b^2 + 2k(n-k)c^2)(1-(2p-1)^2) = S(\rho)(4p-4p^2). \tag{4}$$

Before the first application of the query transformation, the Frobenius norm is 1. Each further application of the query transformation decreases the Frobenius norm. We have proved that the Frobenius norm has a limit of $\frac{1}{\sqrt{k+1}}$ (Frobenius norm of the limiting state ρ_{lim}). Thus, total decrease of the Frobenius norm is $1 - \frac{1}{\sqrt{k+1}}$. Similarly, the square of the Frobenius norm decreases from 1 to $\frac{1}{k+1}$ and has the total decrease of $\frac{k}{k+1}$.

Among first $2m$ applications of the query transformation, there exist two sequential applications which decrease the square of the Frobenius norm by less than $\frac{1}{m}$. Let ρ_1 and ρ_2 be density matrices before these applications. Let a_1, b_1, c_1, d_1 and a_2, b_2, c_2, d_2 be a, b, c, d values of ρ_1 and ρ_2 respectively.

From (4) we have

$$S(\rho_1) < \frac{1}{m(4p-4p^2)} \quad \text{and} \quad S(\rho_2) < \frac{1}{m(4p-4p^2)}. \tag{5}$$

In the further proof we use the following straightforward-to-prove lemma:

Lemma 1 *If $S = k(k-1)b^2 + 2k(n-k)c^2 < R$ and $k \geq 2$ hold then $|c| < \sqrt{\frac{R}{n}}$ and $|b| < \sqrt{R}$ also hold.*

We also use the notation $\delta(a,b) = \{x | a - b < x < a + b\}$.

Lemma 1 and the equation (5) implies

$$c_1 \in \delta\left(0, \sqrt{\frac{R}{n}}\right),$$

$$b_1 \in \delta\left(0, \sqrt{R}\right),$$

$$c_2 \in \delta\left(0, \sqrt{\frac{R}{n}}\right),$$

$$b_2 \in \delta\left(0, \sqrt{R}\right),$$

where $R = \frac{1}{m(4p-4p^2)}$.

The diffusion matrix changes each element a of a vector to $2v - a$, where v is the arithmetic mean of all elements. We will call this the diffusion matrix property.

The arithmetic mean of each of the first k columns of the matrix ρ'_1 (after the first application of the query transformation) is

$$v \in \delta\left(\frac{a_1}{n}, \sqrt{R}\frac{k-1}{n} + \sqrt{\frac{R}{n}}\frac{n-k}{n}\right) \subseteq \delta\left(\frac{a_1}{n}, \frac{k}{n}\sqrt{R} + \sqrt{\frac{R}{n}}\right).$$

Because of the diffusion matrix property, the value of the last elements of the first k columns of the matrix $D\rho'_1$ is

$$c'_1 = 2v - c_1 \in \delta\left(2\frac{a_1}{n}, \frac{2k}{n}\sqrt{R} + 3\sqrt{\frac{R}{n}}\right).$$

The arithmetic mean of each of the last $n-k$ columns of the matrix ρ'_1 is

$$v \in \delta\left(d_1\frac{n-k}{n}, \frac{k}{n}\sqrt{\frac{R}{n}}\right).$$

Hence, the value of the last elements of the last $n-k$ columns of the matrix $D\rho'_1$ is

$$d'_1 = 2v - d_1 \in \delta\left(d_1\frac{n-2k}{n}, \frac{2k}{n}\sqrt{\frac{R}{n}}\right).$$

The arithmetic mean of the last row of the matrix $D\rho'_1$ is

$$v \in \delta\left(a_1\frac{2k}{n^2} + d_1\frac{(n-k)(n-2k)}{n^2}, \frac{2k^2}{n^2}\sqrt{R} + \frac{5nk - 2k^2}{n^2}\sqrt{\frac{R}{n}}\right).$$

Assuming $n > 2k$ and using the definition of the diffusion matrix, we obtain

$$c_2 = 2v - c_1' \in$$

$$\in \delta\left(-2a_1\frac{n-2k}{n^2} + 2d_1\frac{(n-k)(n-2k)}{n^2}, \frac{4k^2}{n^2}\sqrt{R} + \frac{10nk-4k^2}{n^2}\sqrt{\frac{R}{n}} + \frac{2k}{n}\sqrt{R} + 3\sqrt{\frac{R}{n}}\right) \subseteq$$

$$\subseteq \delta\left(-2a_1\frac{n-2k}{n^2} + 2d_1\frac{(n-k)(n-2k)}{n^2}, \frac{4k}{n}\sqrt{R} + 13\sqrt{\frac{R}{n}}\right) =$$

$$= \delta\left(\frac{2(d_1(n-k)-a_1)(n-2k)}{n^2}, \frac{4k}{n}\sqrt{R} + 13\sqrt{\frac{R}{n}}\right).$$

As $c_2 \in \delta\left(0, \sqrt{\frac{R}{n}}\right)$, $\left|\frac{2(d_1(n-k)-a_1)(n-2k)}{n^2}\right| < \frac{4k}{n}\sqrt{R} + 14\sqrt{\frac{R}{n}}$ holds.

As $ka_1 + d_1(n-k) = 1$, it follows that $d_1(n-k) - a_1 = 1 - (k+1)a_1$. Using the inequality

$$\left|\frac{k}{k+1} - ka_1\right| < |1-(k+1)a_1|,$$

we obtain

$$\left|\frac{k}{k+1} - ka_1\right| < \left(\frac{2k}{n} + \frac{7}{\sqrt{n}}\right)\frac{n^2}{n-2k}\sqrt{R}.$$

The left side of this inequality is the absolute value of the difference between the probability of finding any of the marked elements and $\frac{k}{k+1}$.

For an arbitrary ϵ the inequality

$$\left(\frac{2k}{n} + \frac{7}{\sqrt{n}}\right)\frac{n^2}{n-2k}\sqrt{R} < \epsilon$$

holds if

$$m > \frac{1}{4p(1-p)\epsilon^2}\left(\frac{2k}{n} + \frac{7}{\sqrt{n}}\right)^2\frac{n^4}{(n-2k)^2} = O(n)$$

(substituting $R = \frac{1}{4mp(1-p)}$). □

References

1. Ambainis, A.: Quantum search algorithms. SIGACT News 35(2), 22–35 (2004)
2. Buhrman, H., Newman, I., Röhrig, H., de Wolf, R.: Robust Polynomials and Quantum Algorithms. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 593–604. Springer, Heidelberg (2005)
3. Grover, L.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th ACM STOC, Philadelphia, Pennsylvania, pp. 212–219. ACM Press (1996)
4. Horn, R., Johnson, C.: Matrix Analysis. Cambridge University Press (2006)

5. Long, G.L., Li, Y.S., Zhang, W.L., Tu, C.C.: An intrinsic limitation on the size of quantum database. Physical Review A 61, 042305 (2000); Also arXiv:quant-ph/9910076
6. Kaye, P., Laflamme, R., Mosca, M.: An Introduction to Quantum Computing. Cambridge University Press (2007)
7. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
8. Regev, O., Schiff, L.: Impossibility of a Quantum Speed-Up with a Faulty Oracle. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 773–781. Springer, Heidelberg (2008)
9. Shapira, D., Mozes, S., Biham, O.: The effect of unitary noise on Grover's quantum search algorithm. Physical Review A 67, 042301 (2003); Also arXiv:quant-ph/0307142
10. Shenvi, N., Brown, K.R., Whaley, K.B.: Effects of Noisy Oracle on Search Algorithm Complexity. Physical Review A 68, 052313 (2003); Also quant-ph/0304138

On WQO Property for Different Quasi Orderings of the Set of Permutations

Sandra Ose and Juris Viksna

Institute of Mathematics and Computer Science, University of Latvia,
Rainis boulevard 29, Riga LV 1459, Latvia
sandra.ose.z@gmail.com, juris.viksna@lumii.lv

Abstract. The property of certain sets being well quasi ordered (WQO) has several useful applications in computer science – it can be used to prove the existence of efficient algorithms and also in certain cases to prove that a specific algorithm terminates.

One of such sets of interest is the set of permutations. The fact that the set of permutations is not WQO has been rediscovered several times and a number of different permutation antichains have been published. However these results apply to a specific ordering relation of permutations $\preccurlyeq$, which is not the only 'natural' option and an alternative ordering relation of permutations $\trianglelefteq$ (more related to 'graph' instead of 'sorting' properties of permutations) is often of larger practical interest. It turns out that the known examples of antichains for the ordering $\preccurlyeq$ can't be used directly to establish that $\trianglelefteq$ is not WQO.

In this paper we study this alternative ordering relation of permutations $\trianglelefteq$ and give an example of an antichain with respect to this ordering, thus showing that $\trianglelefteq$ is not WQO. In general antichains for $\trianglelefteq$ cannot be directly constructed from antichains for $\preccurlyeq$, however the opposite is the case – any antichain for $\trianglelefteq$ allows to construct an antichain for $\preccurlyeq$.

1 Introduction

The property of certain sets being *well quasi ordered* (WQO) has a number of well known and useful applications in computer science. Firstly, it can be used as a very powerful tool to prove that for a specific problem efficient algorithms exist (for a comprehensive overview see [5]). The most widely used WQO for proving that an efficient algorithm exists is *graph minor* ordering. The fact that this ordering is WQO is highly nontrivial, the proof has been obtained by N. Robertson and P. Seymour and published in a series of more than 20 papers [14]. They also have shown that for any (fixed) graph H there is an algorithm for checking whether H is a minor of G in time polynomial of size of G.

However assuming these facts as known we have a 'trivial' proof that there is a polynomial time algorithm e.g. for Graph Planarity problem – it is sufficient just to note that a minor of planar graph is also planar (in this particular case it is even possible to obtain a concrete, albeit completely impractical, polynomial algorithm by using the method of self-reduction [6]; whilst in general WQO based arguments can give just a nonconstructive proof of algorithm's existence). Most often however WQO based proofs are used to show the existence of parameterized algorithms – e.g. Graph Planarity can be viewed as a sub-case of Graph Genus problem (embedding graph on a surface of genus k) with a particular value of parameter $k = 0$; for any fixed k the same argument shows the existence of a polynomial algorithm for the corresponding Graph Genus problem.

Another useful WQO is *immersion ordering* (also shown to be WQO by the authors of [14]). For certain graph problems (e.g. Graph Cutwidth) it can provide at least simpler proofs of algorithm's existence than graph minor ordering. One can also note that *substring ordering* of finite strings (the notion of well quasi orderings was introduced in [8], where WQO property of finite strings was shown by Higman's Lemma) provides an easy proof of existence of parameterized algorithm for the Closest String problem (a practical algorithm for the problem was given by J. Gramm *et al.* in [7]). There is an ongoing research of orderings that in general are not WQO, but hold such a property for specific classes of sets or graphs and we expect that a number of them could be useful for providing algorithm existence proofs.

Although WQO based approaches never provide an algorithm of any use in practice, the knowledge that an efficient algorithm exists provides good motivation for designing a practical one. Whilst there seem to be some discussion [11] that for some problems we might have only algorithms that are good asymptotically, but not of practical use, in most cases however attempts to design practical algorithms seem to be fruitful.

Another important application of well quasi orderings in computer science is the use of WQO property to prove that a certain algorithm terminates. This is a somewhat more specialized area and the method applies to algorithms used for symbolic verification of infinite state transition systems (lossy channel systems, Petri nets, relational automata etc.); the WQO property here is used to prove that the state backward reachability problem is algorithmically decidable. A comprehensive survey of these methods is given by P. Abdulla and B. Jonsson [4], the authors have also some very recent publications [3,1] explicitly focused on application of different well quasi orderings for analysis of transition systems.

One of the first WQOs applied in this area is *substring ordering* of finite strings [2], but there are many WQOs of interest [1], for example the ordering of powersets, which is shown to be WQO if and only if the initial sets do not contain Rado structures [9,13]. Our interest of WQO property of the particular ordering of the set of permutations also stems from applications in verification of transition systems.

The fact that the set of permutations is not well quasi ordered has been known since early 70-s, the first examples of infinite antichains of permutations being published by R. Tarjan [16] and V. Pratt [12] and according to surveys such examples have been known even earlier [10]. These early results however are not stated in very explicit form, leading to several rediscoveries of this fact – a very recent example of an infinite antichain of permutations has been published as recently as in 2000 by D. Spielman and M. Bona [15]. This appears to be the only published example that can be easily traced by a non-expert without doing deeper research in the topic.

As already discussed one of the reasons for being interested in well quasi orderings of sets are applications of WQO property in computer science. From the perspective of such applications of significance is WQO property of the set of permutations with respect to an ordering relation (denoted here by $\trianglelefteq$), which is different from the ordering relation (denoted here by $\preccurlyeq$) used in published examples referenced above.

Hence it turns out that there are at least two natural definitions of a quasi ordering relation in the set of permutations. The relations $\preccurlyeq$ and $\trianglelefteq$ generally are incomparable, thus the fact that permutations are not WQO with respect to $\preccurlyeq$ does not provide an answer whether permutations are WQO with respect to $\trianglelefteq$. However it turns out that there also exists an infinite antichain of permutations with respect to the ordering relation $\trianglelefteq$, therefore proving that the set of permutations is not WQO with respect to $\trianglelefteq$.

In this paper we give an example of such an antichain. Although some ideas for construction of this antichain have been borrowed from the published examples cited above, these antichains cannot be at least directly transformed to antichains for $\trianglelefteq$ relation. On the other hand, any example of antichain for $\trianglelefteq$ relation, although not being an antichain for $\preccurlyeq$ itself, easily allows to construct an antichain that works for the $\preccurlyeq$ relation.

2 Quasi Orderings of Permutations

We assume that the reader is familiar with the notion of well quasi orderings and give just very brief definitions: a set X is well quasi ordered (WQO) with respect to a quasi ordering $\leq$ if for any infinite sequence of X elements $x_1, x_2, x_3, \ldots$ there exist such indexes i and j that $i < j$ and $x_i \leq x_j$. It is a well known result that this definition is equivalent with the statement that there does not exist an infinite sequence of X elements $x_1, x_2, x_3, \ldots$ such that $x_i \not\leq x_j$ and $x_j \not\leq x_i$ for all $i \neq j$ (a sequence with such a property is called an *antichain*).

We define a permutation p of length n as a bijective mapping $p : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. There are two widely used notations for permutations: most frequently a permutation p of length n is written using so called *one-line notation* as $[p(1), p(2), \ldots, p(n)]$, however a *cycle notation* is also often used. A cycle in a permutation is a sequence $x_1, x_2, \ldots, x_k$ of distinct elements from $\{1, 2, \ldots, n\}$ such that $p(x_k) = x_1$ and $p(x_i) = x_{i+1}$ for all $i = 1 \ldots k - 1$. Each permutation can be decomposed into one or several disjoint cycles and can be written as $p = (x_{1_1}, x_{1_2}, \cdots, x_{1_{k_1}}) \cdots (x_{l_1}, x_{l_2}, \cdots, x_{l_{k_l}})$. Permutation is *cyclic* if it consists of a single cycle. For convenience, unless stated otherwise, we will further consider only cyclic permutations (since our antichain example contains only cyclic permutations whilst obviously serving also as an antichain example for the whole set of permutations). It is easy to see that each permutation has only one representation in one-line notation, but n different cyclic notations (where n is permutation's length): e.g. permutation $[3, 4, 2, 1]$ in cyclic notation can be written as $(1, 3, 2, 4)$,

$(3, 2, 4, 1)$, $(2, 4, 1, 3)$ or $(4, 1, 3, 2)$. (The seeming redundancy of one-line notation is compensated by the fact that not all permutations are cyclic, e.g. a cyclic notation for $[2, 1, 4, 3]$ is $(1, 2)(3, 4)$.)

The only quasi ordering relation of permutations $\preccurlyeq$ that we know to have been considered previously is defined in terms of one-line notation as follows.

Definition 1. *Let p and q be two permutations of length respectively m and n. Then $p \preccurlyeq q$ iff there exists an injective mapping $\phi : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, n\}$ such that for all $i, j \in \{1, 2, \ldots, m\}$:*

1. *$\phi(i) < \phi(j)$ iff $i < j$,*
2. *$p(i) < p(j)$ iff $q(\phi(i)) < q(\phi(j))$.*

For example $[2, 3, 1] \preccurlyeq [3, 4, 2, 1]$ since the mapping $\phi(1) = 1$, $\phi(2) = 2$, $\phi(3) = 4$ satisfies the required properties. If $p \preccurlyeq q$ permutation p is sometimes called a *pattern* of permutation q.

It is easy to see that the relation $\preccurlyeq$ defines a quasi ordering in the set of permutations (the relation is both transitive and reflexive). Informally we can think of pattern of a permutation as of permutation obtained by removal of some elements from the set $\{1, 2, \ldots, n\}$ and by renumbering the remaining elements with consecutive integers starting from 1 in such a way that the order of elements is preserved.

This ordering relation is very natural if we regard permutations as reorderings of elements of some ordered set S – given $p \preccurlyeq q$, permutation p corresponds to a reordering, which is consistent with reordering of q for a particular subset of S. However the relation $\preccurlyeq$ generally does not preserve cycle structures of permutations, for example $[1, 2] \preccurlyeq [2, 3, 1]$, whilst $[2, 3, 1]$ is cyclic but $[1, 2]$ is not.

Nevertheless, sometimes it is much more natural to require that cycle structure should be preserved by an ordering relation. This probably can be best illustrated by regarding permutations as vertex-ordered graphs with vertex set $\{1, 2, \ldots, n\}$ and each vertex having exactly one incoming and exactly one outgoing edge. In such a representation we would like to consider permutation p to be 'smaller' than q if it can be obtained by contraction of a number of q edges (this is probably somewhat similar to a graph minor operation in a very restricted subset of directed graphs). The concept is illustrated in Figure 1.

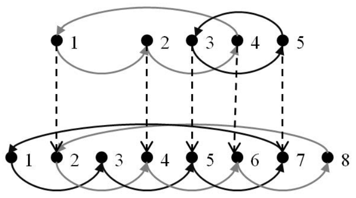


Fig. 1. Ordering relation $\trianglelefteq$ preserves cycle structure

Let p be a permutation of length n. We say that (x, y) is an *edge* of p if $p(x) = y$ and we say that the sequence $P(x_1, x_k) = x_1, \ldots, x_k$ is a *path* of length k in p if $p(x_i) = p(x_{i+1})$ for all $i = 1 \ldots k-1$. Using this notation we can formally define the ordering relation described above as follows.

Definition 2. *Let p and q be two permutations of length respectively m and n. Then $p \trianglelefteq q$ iff there exists an injective mapping $\psi : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, n\}$ such that for all $i, j \in \{1, 2, \ldots, m\}$:*

1. *$\psi(i) < \psi(j)$ iff $i < j$,*
2. *(i, j) is an edge in p iff there exists a path $x_1 = \psi(i), x_2 \ldots, x_{k-1}, x_k = \psi(j)$ in q such that for all $s = 2 \ldots k-1$ there does not exist y with $\psi(y) = x_s$.*

Although injective mappings in both definitions look similar, we have used different letters on purpose. The mapping ϕ has to be thought of as a function that maps *positions* in sequences representing permutations, whilst ψ has to be thought of as a function that maps *values* of these sequences.

For example $(1, 2, 7, 8, 5, 6, 3, 4) \trianglelefteq (1, 2, 11, 12, 9, 10, 7, 8, 5, 6, 3, 4)$ (define $\psi(1) = 1, \psi(2) = 2$ and $\psi(i) = i + 4$ otherwise).

However $(1, 2, 7, 8, 5, 6, 3, 4) \not\preccurlyeq (1, 2, 11, 12, 9, 10, 7, 8, 5, 6, 3, 4)$ (in one-line notation these are permutations $[2, 7, 4, 1, 6, 3, 8, 5]$, $[2, 11, 4, 1, 6, 3, 8, 5, 10, 7, 12, 9]$ and actually come from the Tarjan's antichain example for $\preccurlyeq$). If $p \trianglelefteq q$ we will say that p is a *subpermutation* of q.

We have already seen that $[1, 2] \preccurlyeq [2, 3, 1]$ and $[1, 2] \not\trianglelefteq [2, 3, 1]$, thus in general the ordering relations $\preccurlyeq$ and $\trianglelefteq$ are incomparable (but we do not know whether there are *cyclic* permutations for which holds $\preccurlyeq$, but not $\trianglelefteq$; our antichain for $\trianglelefteq$ can be shown to be also an antichain for $\preccurlyeq$).

However we can look at the definitions for these ordering relations just as at syntactic requirements that some sequences of natural numbers have to satisfy certain properties, without taking care what permutations (if any) these sequences define. From such a perspective it turns out that the relation $\trianglelefteq$ is stronger than $\preccurlyeq$.

Let $C_{p,i}$ be a cyclic notation for permutation p that starts with a number i. If the permutation p has length n then it has n different cyclic notations $C_{p,1}, \ldots, C_{p,n}$. For a given cyclic notation $C = (y_1, \ldots, y_n)$ let $C(i)$ denote its i-th element y_i. For permutation of length n and its cyclic notation C by $P(C)$ we denote the permutation $[C(1), \ldots, C(n)]$ (i.e. we assume that the sequence of numbers in cyclic notation actually describes a permutation in one-line notation).

Theorem 1. *Let p and q be two permutations of length respectively m and n. Then $p \trianglelefteq q$ iff for each $i \in \{1, \ldots, m\}$ there exists $j \in \{1, \ldots, n\}$ such that $P(C_{p,i}) \preccurlyeq P(C_{q,j})$.*

Proof. ($\Rightarrow$). Let $p \trianglelefteq q$, let ψ be a mapping that validates this relation and let $i \in \{1, \ldots, m\}$. We consider $C_p = C_{p,i}$, $C_q = C_{q,\psi(i)}$ and will show that $P(C_p) \preccurlyeq P(C_q)$ by defining ϕ recursively as follows: $\phi(1) = 1$, $\phi(k) = \phi(k-1) + \mathit{length}(\psi(C_p(k-1)), \psi(C_p(k)))$ for $k > 1$, where $\mathit{length}(\psi(C_p(k-1)), \psi(C_p(k)))$ denotes the length of path between $\psi(C_p(k-1))$ and $\psi(C_p(k))$ in C_q.

Since $\phi(1) = 1$ and $length(\psi(C_p(k-1)), \psi(C_p(k))) \geq 1$ for all $k > 1$ we have $\phi(k) < \phi(l)$ for all $k < l$, i.e. property 1 of Definition 1 holds.

We need to prove the second property that $C_p(k) < C_p(l)$ iff $C_q(\phi(k)) < C_q(\phi(l))$. Notice that $\psi(C_p(1)) = \psi(i) = C_q(1) = C_q(\phi(i))$ and from a simple proof by induction it follows that $\psi(C_p(k)) = C_q(\phi(k))$ (actually the implicit motivation behind our choice of ϕ is to ensure that this property holds). Therefore $C_p(k) < C_p(l)$ iff $\psi(C_p(k)) < \psi(C_p(l))$ (by the definition of ψ), and $C_p(k) < C_p(l)$ iff $C_q(\phi(k)) = \psi(C_p(k)) < C_q(\phi(l)) = \psi(C_p(l))$ Q.E.D.

($\Leftarrow$). Let $P(C_p) \preccurlyeq P(C_q)$, where $C_p = C_{p,i}$, $C_q = C_{q,j}$ for some i and j, and let ϕ be a mapping that validates this relation. We will use the mapping $\psi(C_p(k)) = C_q(\phi(k))$ to show that $p \trianglelefteq q$ holds (since $\{C_p(k)|k = 1, \ldots, m\} = \{1, \ldots, m\}$ the mapping ψ is completely defined).

Due to the second property of Definition 1 $C_p(k) < C_p(l)$ iff $C_q(\phi(k)) < C_q(\phi(l)))$ and thus $C_p(k) < C_p(l)$ iff $\psi(C_p(k)) = C_q(\phi(k)) < \psi(C_p(l)) = C_q(\phi(l))$, i.e. property 1 of Definition 2 holds.

We need to show that property 2 from Definition 2 holds for all edges of p. There are m such edges: $(C_p(k), C_p(k+1))$ for $k = 1 \ldots m-1$ and an edge $(C_p(m), C_p(1))$.

Consider an edge $(C_p(k), C_p(k+1))$ in p. Then for $\psi(C_p(k)) = C_q(\phi(k)) = C_q(s)$ and $\psi(C_p(k+1)) = C_q(\phi(k)) = C_q(t)$ we should have $s < t$ (due to property 1 of Definition 1) and by the definition of cyclic permutation there is a path $C_q(s) = \psi(C_p(k)), C_q(s+1), \ldots, C_q(t) = \psi(C_p(k+1))$ in q (notice that in cyclic notation all elements of a path should appear in consecutive positions).

We have to show that for all $x = s+1, \ldots, t-1$ there are no ψ preimages for $C_q(x)$. Assume this is not the case and $C_q(x) = \psi(C_p(y))$ for some $x \in \{s+1, \ldots, t-1\}$ and $y \in \{1, 2, \ldots, m\}$.

Since $\psi(C_p(k)) = C_q(s)$ and $\psi(C_p(k+1)) = C_q(t)$ we should have either $y < k$ or $y > k+1$. If $y < k$ we will have $C_q(x) = \psi(C_p(y)) = C_q(\phi(y)) < C_q(s) = C_q(\phi(k))$ and due to property 2 of Definition 1 $x < s$ contradicting $x \in \{s+1, \ldots t-1\}$. In an analogous way for $y > k+1$ we obtain $x > t$ again contradicting $x \in \{s+1, \ldots t-1\}$.

For the remaining edge $(C_p(m), C_p(1))$ in a similar way we can show that there are no preimages for q elements in positions smaller than s or larger than t, where $C_q(s) = \psi(C_p(1))$ and $C_q(t) = \psi(C_p(m))$.

Whilst the proof is notationally somewhat complicated, the motivating idea beyond the result of Theorem 1 is simple and is informally illustrated in Figure 2.

Finally, note that $p \trianglelefteq q$ does not require that $P(C_{p,i}) \preccurlyeq P(C_{q,j})$ for all cyclic representations of p and q, e.g. $(1, 2, 3) \trianglelefteq (2, 3, 1)$ (these permutations are equal), however $[1, 2, 3] \not\preccurlyeq [2, 3, 1]$.

This result implies that any antichain with respect to the ordering relation $\trianglelefteq$ easily allows to obtain an antichain for the ordering relation $\preccurlyeq$ (for a given antichain $C_1, C_2, \ldots$ simply consider the antichain $P(C_1), P(C_2), \ldots$), however the converse is not necessarily true.

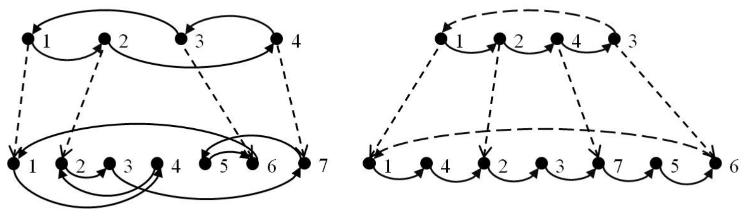


Fig. 2. In cyclic notation permutations 'unfold' and the relation $\trianglelefteq$ between two cyclic permutations can be thought of as a mapping between sets of vertices placed on borders of two circles without any crossings of 'edges' of the mapping. By making a 'cut' that breaks both circles, but not any of mapping's 'edges', we obtain two strings of numbers that can be regarded as permutations ordered by $\preccurlyeq$.

3 Antichain of Permutations for the Ordering $\trianglelefteq$

As can be seen from the previous chapter the antichain example:

$[2, 7, 4, 1, 6, 3, 8, 5]$, $[2, 11, 4, 1, 6, 3, 8, 5, 10, 7, 12, 9]$,

$[2, 15, 4, 1, 6, 3, 8, 5, 10, 7, 12, 9, 14, 11, 16, 13], \dots$

given by R. Tarjan [16] does not work for the ordering relation $\trianglelefteq$. V. Pratt's example [12] is similar. A different construction is given by D. Spielman and M. Bona [15], however it too does not work for the ordering relation $\trianglelefteq$. This also remains true if we simply try to 'interpret' these examples as sets of permutations written in cyclic notation, instead of one-line notation they were intended to be. Still in the antichain example we provide for $\trianglelefteq$ ordering relation we have borrowed some 'patterns' from Spielman's and Bona's construction.

We define the set of permutations $\mathcal{P}$ as follows.

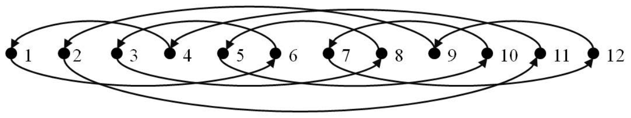
Definition 3. *$\mathcal{P} = \{p_i | i = 5, 6, 7 \dots\}$, where $p_i = (4, 1, 6, \underline{3, 8, \dots}, 2n-5, 2n, 2n-3, 2, 2n-1)$, where the underlined part corresponds to sequence $a_0, b_0, a_1, b_1, \dots, a_k, b_k$ with $a_i = 3 + 2i$, $b_i = 8 + 2i$ and $k \geq 0$.*

The set $\mathcal{P}$ consists of permutations $p_5, p_6, p_7, \dots$ with permutations p_i having length $2i$. For example $p_6 = (4, 1, 6, 3, 8, 5, 10, 7, 12, 9, 2, 11)$. Permutations p_6 and p_7 are shown in Figure 3.

We will prove that the set of permutations $\mathcal{P}$ contains an antichain $p_5, p_6, p_7, p_8, \dots$. We start with two simple lemmas.

Lemma 1. *Let p and q be two permutations of length respectively m and n, such that $p \trianglelefteq q$ holds via an injective mapping ψ. Then for any $a, b \in \{1, \dots, m\}$, $s, t \in \{1, \dots, n\}$, such that $a < b$ and $\psi(a) = s, \psi(b) = t$, the following holds:*

1. *$b - a \leq t - s$,*
2. *if $a = s$, then $\psi(x) = x$ for all $x \leq a$,*
3. *if $m - b = n - t$, then $\psi(x) = x + (n - m)$ for all $x \geq b$,*
4. *if $b - a = t - s$, then $\psi(x) = s + (x - a)$ for all $x = a \dots b$.*



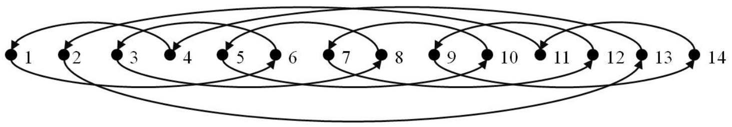


Fig. 3. Permutations p_6 and p_7

Proof. All these are trivial properties of any increasing injective mapping $\psi : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, n\}$.

Lemma 2. *Let p and q be two permutations of length respectively m and n, such that $p \trianglelefteq q$ holds via an injective mapping ψ. Let $a, b \in \{1, \ldots, m\}$, $s, t \in \{1, \ldots, n\}$ and $\psi(a) = s, \psi(b) = t$. If $x_1 = a, x_2, \ldots, x_k = b$ is a path in p and $y_1 = s, y_2, \ldots, y_k = t$ is a path in q then $\psi(x_i) = y_i$ for all $i = 1 \ldots k$.*

Proof. Assume this is not the case and $\psi(x_i) \neq y_i$ for some i. Then there are two possibilities:

1. There exist $i_1 < i_2$ and $j_1 > j_2$ such that $\psi(x_{i_1}) = y_{j_1}$ and $\psi(x_{i_2}) = y_{j_2}$. Then the path $x_{i_2}, \ldots, x_k$, not containing x_{i_1}, should be mapped by ψ to the path $y_{j_2}, \ldots, y_{j_1}, \ldots, y_k$ containing $y_{j_1} = \psi(x_{i_1})$. This contradicts property 2 of Definition 2.
2. There exists i such that $\psi(x_i)$ is mapped to an element z not in $y_1, y_2, \ldots, y_k$. Then the path $x_i \ldots, x_k$, not containing x_1, should be mapped by ψ to the path $z, \ldots, y_1, \ldots, y_k$ containing $y_1 = \psi(x_1)$, again giving a contradiction.

Our main result is a proof that the sequence of permutations $p_5, p_6, p_7, p_8, \ldots$ from $\mathcal{P}$ is an antichain with respect to the ordering relation $\trianglelefteq$.

Theorem 2. *For any $m, n \geq 5$, if $m \neq n$ then permutations $p_m, p_n \in \mathcal{P}$ are incomparable with respect to the ordering $\trianglelefteq$.*

Proof. By definition p_m has length $2m$ and p_n has length $2n$. Without the loss of generality we can assume $m < n$. Then the fact $p_n \not\trianglelefteq p_m$ is trivial.

Otherwise assume the converse holds and $p_m \trianglelefteq p_n$ via mapping ψ. We will try to deduce some constraints on ψ.

Let us consider the value of $\psi(4)$. Due to Lemma 1 we cannot map element 4 to any of the elements $1, 2, 3, 2n-4, 2n-3, 2n-2, 2n-1, 2n$.

Consider the option of mapping 4 to an odd element $2k-1$, where $k = 3 \ldots n-2$. The edge $(4, 1)$ from p_m has to be mapped to a path $y_1 = 2k-1, y_2, \ldots, y_l$ in p_n, such that $y_l = \psi(1) < \psi(4) = 2k-1$ and there are no preimages for elements $y_2, \ldots, y_{l-1}$. The shortest path with such a property has $y_l = 2$ and contains all the elements from the set $\{2k-1, \dots, 2n\}$ except $2k$, $2k+2$ and $2n-1$. Thus ψ should map the set $\{5, 6, \dots, 2m\}$ having at least 6 elements to the set $\{2k, 2k+1, 2n-1\}$ of 3 elements, which is impossible.

Now consider the option of mapping 4 to an even element $2k$, where $k = 3 \dots n-3$. If additionally $\psi(1) = 2k-3$, then also $\psi(2) = 2k-2$ and $\psi(3) = 2k-1$ (by Lemma 1). Then the edge $(2, 2m-1)$ in p_m should be mapped to a path $y_1 = 2k-2, y_2, \dots, y_l$ in p_n such that $y_l > \psi(4) = 2k$ and there are no preimages for elements $y_2, \dots, y_{l-1}$. However all such paths contain an element $y_i = \psi(2k)$ making this impossible.

Alternatively, if $\psi(1) \neq 2k-3$ we should map $(4, 1)$ to a path $y_1 = 2k, y_2, \dots, y_l$ with $y_l = \psi(1) < \psi(4) = 2k$, $\psi(1) \neq 2k-3$ and $\psi(1) \neq 2k-1$ (due to Lemma 1). The shortest path with such a property again has $y_l = 2$ and contains all the elements from the set $\{2k, \dots, 2n\}$ except $2n-1$, making ψ impossible.

Thus we must have $\psi(4) = 4$ as well as $\psi(1) = 1$, $\psi(2) = 2$, $\psi(3) = 3$. We will show that in this case $\psi(i) = i$ for all $i = 1 \dots 2m-4$.

Consider the path $1, 6, 3$ in p_m. Since $\psi(1) = 1, 6, \psi(3) = 3$ is a path in p_n, by Lemma 2 we should have $\psi(6) = 6$ leading to $\psi(5) = 5$ (by Lemma 1). Applying the same argument to path $3, 8, 5$, then to path $5, 10, 7$ etc. we can prove by induction that $\psi(i) = i$ for all $i = 1 \dots 2m-4$

We can apply the reasoning above also by starting from the other 'end' of permutations – firstly, showing that $\psi(2m-3) = 2n-3$ and then that $\psi(i) = i + (2n-2m)$ for all $i = 5, \dots, 2m$.

With all these constraints on ψ what can be the value of $\psi(5)$? Since $2m \geq 10$ the $5 < 2m-4$, and so (see two paragraphs above) $\psi(5) = 5$. But we also have shown that $\psi(5) = 5 + (2n-2m) > 5$ (see the last paragraph). Hence we have a contradiction and the assumption that $p_m \trianglelefteq p_n$ is false.

4 Conclusions

We have to admit that contrary to some initial expectations the result we are presenting here is negative, i.e. the set of permutations is not WQO also for the relation $\trianglelefteq$, thus does not lead to any practical applications. Still we think that this result might be of interest to other researchers, taking into account that we ourselves were quite surprised to find that the 'folklorized' result of permutations not being WQO does not really apply to seemingly the most natural ordering relation we were considering.

On a broader perspective, there appears to be certain lack of good surveys about WQO and/or non-WQO sets, which might be of direct interest to computer science as well as about the problems to which WQO based proof techniques can be applied. E.g. the fact that Higman's Lemma gives an easy proof of the existence of a parameterized algorithm Closest String is very simple and probably known by many; however whilst this application may not be unique, we are not aware of any published results regarding the lemma's applications to algorithm design for string related problems.

Acknowledgments. The authors would like to thank the anonymous reviewer #1 for suggesting improved formulation of statement and proof of Theorem 1.

The research was partially supported by Latvian Council of Science grant 09.1578.

References

1. Abdulla, P.: Well (and better) quasi-ordered transition systems. Bulletin of Symbolic Logic 16(4), 457–515 (2010)
2. Abdulla, P., Cerans, K., Jonsson, B., Tsay, Y.: General decidability theorems for infinite-state systems. In: Proceedings of 11th Annual IEEE Symposium on Logic in Computer Science, LICS 1989, pp. 313–321 (1996)
3. Abdulla, P., Chen, Y.-F., Holík, L., Mayr, R., Vojnar, T.: When Simulation Meets Antichains. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 158–174. Springer, Heidelberg (2010)
4. Abdulla, P., Jonsson, B.: Ensuring completeness of symbolic verification methods for infinite-state systems. Theoretical Computer Science 256(1-2), 145–167 (2001)
5. Downey, R., Fellows, M.: Parameterized complexity. Springer (1999)
6. Fellows, M., Langston, M.: On search, decision and the efficiency of polynomial-time algorithms. In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing, STOC 1989, pp. 501–512 (1989)
7. Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica 37, 25–42 (2003)
8. Higman, G.: Ordering by divisibility in abstract algebras. Proceedings of London Mathematical Society 2(7), 326–336 (1952)
9. Jancar, P.: A note on well quasi-orderings for powersets. Information Processing Letters 72 (1999)
10. Jenkyns, T., Nash-Williams, C.: Counter-examples in the theory of well-quasi-ordered sets, Mimeographed (1968)
11. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford University Press (2006)
12. Pratt, V.: Computing permutations with double-ended queues, parallel stacks and parallel queues. In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, STOC 1973, pp. 268–277 (1973)
13. Rado, R.: Partial well-ordering of sets of vectors. Mathematika 1, 89–95 (1954)
14. Robertson, N., Seymour, P.: Graph minors. XX. wagner's conjecture. JCTB: Journal of Combinatorial Theory, Series B 92, 2004
15. Spielman, D., Bóna, M.: An infinite antichain of permutations. The Electronic Journal of Combinatorics 7(N2) (2000)
16. Tarjan, R.: Sorting using networks of queues and stacks. Journal of the ACM 19(2), 341–346 (1972)

Towards User-Aware Multi-touch Interaction Layer for Group Collaborative Systems

Vít Rusňák[1], Lukáš Ručka[1], and Petr Holub[2,3]

[1] Faculty of Informatics
[2] Institute of Computer Science,
Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic
[3] CESNET z.s.p.o., Zikova 4, 162 00 Prague, Czech Republic
{xrusnak,xrucka}@fi.muni.cz, hopet@ics.muni.cz

Abstract. State-of-the-art collaborative workspaces are represented either by large tabletops or wall-sized interactive displays. Extending bare multi-touch capability with metadata for association of touch events to individual users could significantly improve collaborative work of co-located group. In this paper, we present several techniques which enable development of such interactive environments. First, we describe an algorithm for scalable coupling of multiple touch sensors and a method allowing association of touch events with users. Further, we briefly discuss the Multi-Sensor (MUSE) framework which utilizes the two techniques and allows rapid development of touch-based user interface. Finally, we discuss the preliminary results of the prototype implementation.

1 Introduction

Recent advances in multi-touch technology promoted the development of medium and large-sized interactive surfaces both horizontal (i.e., tabletops) and vertical (interactive display walls). Opposed to personal devices such as smart-phones or tablets, these are suitable for multi-user co-located collaboration with large-scale data [1] – e.g., geospatial imagery, scientific visualizations. While association of input actions to individual users could significantly improve a way of interaction [2], only limited attention has been paid to this problem so far.

We focus on utilisation of form-factor multi-touch monitor overlays or monitors with multi-touch sensor embedded to enable direct touch-based interaction in large-sized interactive systems. Autonomous overlay sensors are able only to recognize touch points but cannot associate them with users. We integrate gaming depth sensors such as MS Kinect as additional source of input data to address this issue. The integration of various sensor devices (e.g., touch sensors and depth sensors) opens a range of possibilities to enhance existing multi-touch systems with context-aware and proxemic interaction with only low additional costs.

The rest of the paper is structured as follows. Section 2 reviews related work. In Section 3 we present several techniques enabling *a)* coupling of multi-touch overlay panels which behave as a single seamless surface, *b)* a method to distinguish users and associate them with operations they performed using low-priced off-the-shelf devices. Brief description of Multi-Sensor (MUSE) Framework which implements these techniques is in Section 4. The evaluation methodology and results of the prototype implementation are discussed in Section 5. Section 6 summarizes the findings and outlines future work.

2 Related Work

In this section, we address research topics related to touch-based user interaction augmented with association of input actions with users.

The most used technologies behind the touch-based surface are capacitive and resistive sensors and systems based on IR illumination such as FTIR (Frustrated Total Internal Reflection) or DI (Diffused Illumination) [3]. Some of the technologies become commercially available in a form of overlay frames. For instance, Cyber-Commons [1] uses large multi-touch overlay frame from PQLabs providing up to 32 concurrent touch points allowing multi-user interaction. Although multi-touch technologies enable multi-user capability, none of them are able to provide user tracking feature by themselves.

Interaction spaces [4] installation combines touch-less gesture recognition implemented by a camera array placed along the display wall with acoustic control for windows positioning triggered via snapping. DiamondTouch [5] uses capacitive coupling through the users touching the sensor. Both the systems require users to stay at the same place to distinguish users. HandsDown technique [6] allows user identification and further tracking based on hand contour analysis. However, when a hand of a user leaves the surface area, the tracking is lost. Inexpensive proximity sensors placed around the tabletop [7,8] enable continuous user tracking around and above a tabletop. The main weakness is their vulnerability to reflective materials, resulting in reporting false positives. Another weakness is the recognition precision, if two users overlap hands over the same area and one of them touches the surface, the wrong user could be associated [8]. Bootstrapper [9] project utilises MS Kinect to distinguish users according to their shoes. This requires them to wear still the same shoes. LightSpace [10] is a small room installation where up to 6 users are able to interact between several uninstrumented surfaces (standard office desk or projection screen). User tracking is realized by three depth cameras. Although multi-touch capability is basically supported on level of hand tracking, finger tracking is hardly achievable due to low resolution of depth sensors. WILD Room project [11] and CGLXTouch [12] represent multi-surface environments where several users can work using multiple hand-held devices (e.g., tablets and cell-phones) and tabletop. Direct Interaction Manager (DIM) [13], an extension to the SAGE middleware [14], utilises gyromouse and Nintendo Wii Remote as pointing devices.

Several frameworks and tools enabling rapid prototyping of multi-user systems with touch interface have appeared recently. ReacTIVision [15] is a computer vision toolkit for development of table-based multi-touch interactive surfaces based on DI and FTIR technologies. EBITA [16] and T3 [17] are software toolkits for development of tiled tabletops and their applications. Component-based framework OpenInterface [18] for prototyping and building multi-modal user interfaces from commodity devices (e.g., Nintendo Wii Remote or iPhone). It is focused on single-user applications of two categories: navigation in large information spaces (e.g., maps) and gaming. More general InTml [19] is a XML-based tool for description of applications with different types of input and output devices, and 3D interaction techniques. It is suitable only for high-level design of interactive system.

Discussed group collaborative systems just pointed out possible research directions. We explore possibility of combining less precise IR camera based user tracking with very accurate data from multi-touch surface to enable such association. We utilise off-the-shelf touch overlay sensors together with gaming depth sensors (e.g., MS Kinect) which can provide more accurate image data than simple proximity sensors. We envision building an environment where users can start working without any obtrusive orchestration or obscure initialization procedure.

3 Design Patterns for Building Interactive Systems from Commodity HW

In this section, we propose design patterns that we considered for building affordable interactive display walls and tabletops. First, we discuss coupling method for multi-touch overlay panels. Then, we describe the user tracking using low-cost depth sensors suitable for tabletop setups. Last, but not least, the approach for association of touch input events with user's hand is presented.

3.1 Virtual Sensor

The virtual sensor concept allows coupling of individual overlay touch panels in order to emulate single seamless multi-touch surface. The underlying time synchronisation and gesture concatenation are hidden, which enable to use existing gesture recognition algorithms in further processing. The algorithm consists of two operations: *a)* remapping of touch events from received messages to dimensions of virtual sensor, and *b)* concatenation of those touch events which are performed over two or more overlay panels as a single stroke.

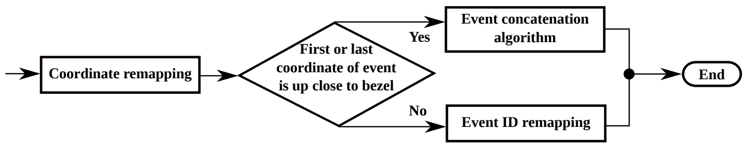

Coordinate remapping
First or last coordinate of event is up close to bezel
Yes
No
Event concatenation algorithm
Event ID remapping
End

Fig. 1. Top-level block diagram of virtual sensor algorithm

Block diagram of the algorithm is listed in Fig. 1. Each message represents a touch event (i.e., touch point or continuous stroke). The message has five parameters – event ID, identification of overlay panel on which the event was performed (*deviceID*), *startTime* and *endTime* event time stamps and a list of coordinates of the touch sensor where the event occurred (*position*). Parameters of each overlay sensor – such as physical dimensions, resolution, width of bezels and position relative to other panels – are stored in a separate configuration file. These parameters describe also virtual sensor properties given by merged dimensions of individual physical sensor panels.

When received a touch event message which starts and ends not too close to bezels (i.e., there is still some distance from the touch point to the bezel) the coordinates of the touch event are remapped according to dimensions of the virtual sensor. This distance is derived from physical parameters of given sensor. Events which end or begin close to the surface bezel are considered as parts of cross-sensor stroke and marked for further processing in concatenation algorithm. Since a trajectory of finger crossing bezels is inaccurate, the concatenation algorithm estimates position of follow-up stroke using time and spatial thresholds. Δt represents time threshold given in milliseconds. Δs is a diameter of an area in which the gesture might continue on adjacent sensor given in pixels. The concatenation operation creates a new event message from the messages given as its parameters. The new message has event ID and *startTime* identical to the first input message and *endTime* from the last one. *Position* list of the output message is concatenation of *position* lists of all events.

3.2 User Tracking with Commodity Devices

Autonomous touch sensors provide only touch input information. To enable user tracking, it is necessary to combine more information about the workspace and its surrounding – e.g., distinguish users interacting with the system, their positions and what they do. Camera-based depth sensors can track individual users, but lack precision of touch sensors [10].

MS Kinect body-tracking algorithms (e.g., [20]) work well only for front-view setups and require the full body to be visible. In tabletop setups, where the sensor is placed above the surface, such algorithms fail. We propose a *relaxed user tracking algorithm* which overcomes the issue. The main idea is to describe image blobs which represent hands and associate them with the particular user using their skeletal descriptions. The block structure is shown in Fig. 2. The algorithm processes a video stream captured by a depth sensor and returns a list of blobs with their descriptions detected in each frame. Blobs represent body

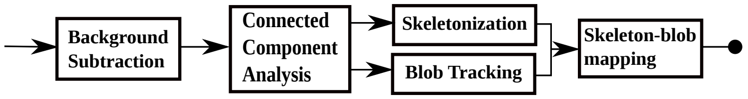


Fig. 2. Relaxed user tracking algorithm

parts of users. They are described by their position, contour, deflection angle, bounding box, type (i.e., right or left hand) and association to user.

First, the background subtraction extracts foreground objects in a previously defined volume in front of touch surface. We deployed the component-labeling algorithm by Chang et al. [21] to detect individual blobs in the frame. Blob tracking in consecutive frames is realized by two-tiered tracking algorithm by Senior et al. [22]. And Zhang-Suen [23] algorithm is adapted to detect 2D skeleton of each blob with supplementary depth parameter. The depth value could be also utilized in further processing to resolve hand occlusions (e.g., user's hand overlaps the hand of another). Skeleton descriptions of blobs are used to construct skeletons of individual users according to their mutual positions and orientations. Fig. 3 illustrates the results generated by blob detection and skeletal detection algorithms executed on the same depth image frame.

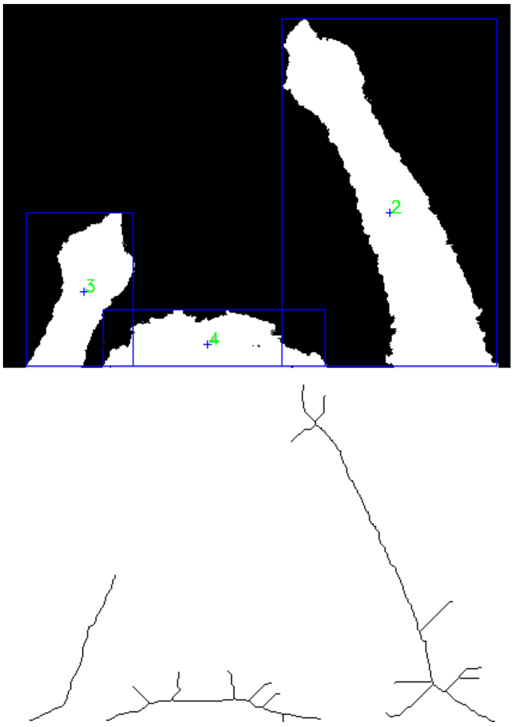


Fig. 3. Blob detection and skeletonization algorithms executed on the same source – from left to right: left hand, part of the head, right hand

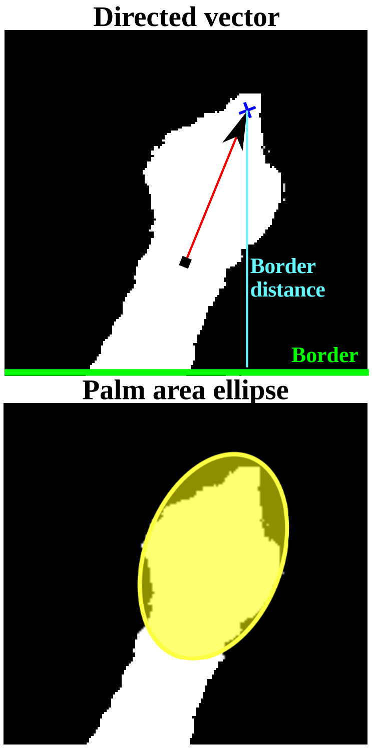


Fig. 4. Visualization of the directed vector and the palm area

3.3 Touch Point Matching

Initial Calibration. Precise mutual calibration of depth sensor and touch surface is crucial for correct matching. Utilizing retro-reflective markers [10] at sensor edges allows easy identification of touch surface position within image frames captured by depth sensor. Moreover, this approach enables automation of calibration procedure.

Touch-point matching algorithm links touch events from virtual sensor with corresponding blobs given by aforementioned user tracking. In the first step, coordinates of touch points from the virtual sensor and position of blobs are transformed to a common corresponding coordinate system. For each blob representing user's hand, the palm area is determined. Due to low-resolution of depth sensors is hard to recognize individual fingers so the algorithm represents the palm as an ellipse. Matching function is simplified to specifying mutual position of touch point and this ellipse defined according to the standard form equation:

$$\frac{(x-m)^2}{a^2} + \frac{(y-n)^2}{b^2} = f(z). \tag{1}$$

The match is recognized when touch point coordinates are at an arbitrary point inside the ellipse ($f(z) < 1$). When touch event is matched with one of the blobs, its description is extended with identification of the corresponding user.

Algorithm 1. Palm Area Detection

```
 1: for all hand-blobs in the frame do
 2:     border := surface border which intersects blob
 3:     for all blob.skeleton.node = leaf node do
 4:         endpoints := endpoints $\bigcup$ (blob.skeleton.node $\bigcap$ surface area)
 5:     end for
 6:     for all endpoints do
 7:         MAX_CRD_DIST := dist(blob.centroid, endpoints.node)
 8:         MAX_BDR_DIST := dist(border, endpoints.node)
 9:         endpoint := node
10:         if (dist(blob.centroid, node) > MAX_CRD_DIST) &&
11: (dist(border, node) > MAX_BDR_DIST) then
12:             endpoint := node
13:         end if
14:     end for
15:     dirVect.start := centroid; dirVect.end := endpoint
16:     blob.palmEllipse := estimateEllipse(dirVect, blob.depth, blob.boundingBox)
17: end for
```

Palm area detection algorithm is shown in Alg. 1. First, a subset of blob skeleton leaf nodes which are within the surface is produced. Then, the endpoint node is selected – the node with maximum distance from *a)* blob centroid (central point of the blob), and *b)* touch surface border which intersects the blob. To determine the palm area, we set a *directed vector* of a blob. The palm ellipse size is estimated from maximum width of the blob along the directed vector. An illustration of the directed vector and palm area are in Fig. 4.

Summary. Virtual sensor algorithm can be used separately to provide an alternative to expensive commercial solutions. By combination of input data from virtual touch sensor and user tracking data (descriptions of blobs) we can create user-aware interactive input layer for tiled display walls or tabletops. The implementation and evaluation of the algorithms is discussed further.

4 MUSE Framework Design

We developed a Multi-Sensor (MUSE) framework as an environment for evaluation of designed algorithms and proposed concept of interactive input layer for high-resolution visualization systems from low-cost touch and depth sensors. The modular structure of the framework allows us to use it as a tool for rapid development of such interactive input layers. In this section we describe its three-layer structure and features.

The communication is supplied with *TUIO 2.0* protocol [24] which provides comprehensive description of touch-based and tangible environments. Fig. 5 shows the framework structure in particular setup that we used for our experiments.

Wrapper Layer. The lowest layer is the interface between various physical sensor devices and unified environment of the MUSE framework. Wrappers convert incoming signals from devices to TUIO 2.0 messages. Each physical device is represented by a single wrapper module. Depending on the type of the input device, wrapper may perform additional processing of the input data. For depth sensor device, the wrapper performs object detection or skeleton tracking to find and distinguish individuals. For touch sensor panel, the wrapper processes coordinates of touch points only.

Virtual Sensor Layer. The middle layer provides the functionality for assembling virtual sensors from sensor wrappers of the same kind. Outgoing message stream from each virtual sensor module contains the information from relevant physical devices but represented as a single seamless sensor. The general idea of virtual sensors is also applicable to other devices than touch sensors (e.g., depth sensors) and we will explore it in future. Also, the modular architecture of the framework allows for an easy development of new virtual sensor modules.

Aggregating Layer. The top layer consists of two main modules – touch-point matching and message generator for output messages. Touch-point matching ensures association of input events from virtual touch sensors to individual users described by linked blobs. Matching algorithm is discussed in Section 3. Finally, output message generator produces outgoing messages which are passed to the application. Messages contain the information about position, where the touch event occurred within the virtual touch sensor and a tag representing association to user who performed it.

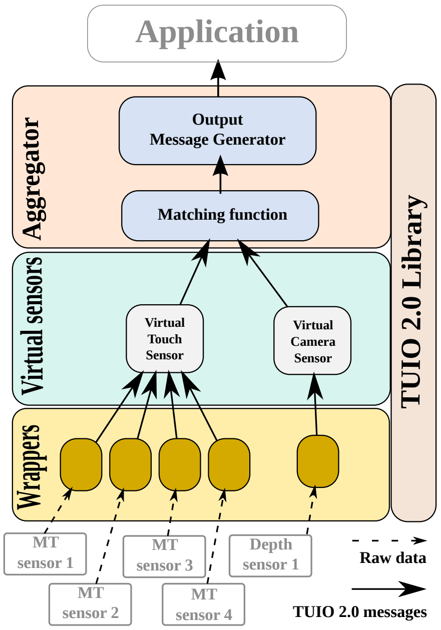


Fig. 5. The MUSE framework diagram for the tiled-tabletop with 4 touch sensors and one depth sensor

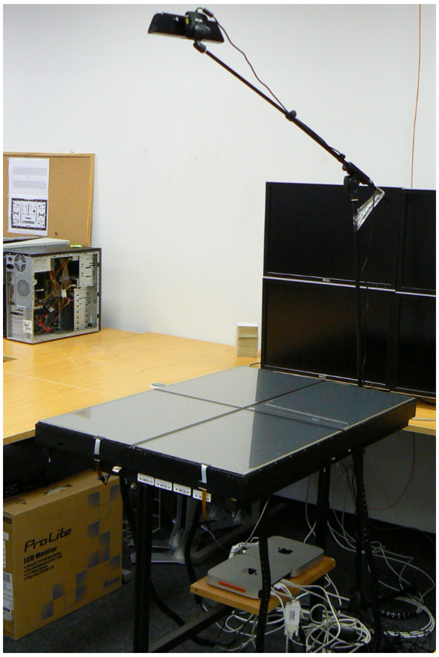


Fig. 6. Prototype of our 2×2 tiled tabletop

5 Experimental Results and Discussion

We constructed a tilting tiled tabletop prototype and developed an initial implementation of the main parts of the MUSE framework – wrappers for touch panels and MS Kinect, virtual sensor and touch point matching modules. The modules were evaluated with focus on their processing speed and accuracy.

5.1 Prototype Implementation

Tabletop Prototype. Our tiled tabletop prototype (see Fig. 6) consists of four 22“ LCDs connected to a single PC running Ubuntu Linux 10.10. We used distributed rendering middleware SAGE for visualization. Each display is equipped with overlay touch sensor panel with frame rate up to 15 Hz. Tabletop was furnished with MS Kinect depth sensor placed above. HW configuration of the desktop PC was as follows: 2× Quad-Core AMD Opteron 2354AM @ 2.2 GHz, 11 GB DDR2 RAM, Gigabit Ethernet, 2× GPU nVidia GeForce 9600 GT.

MUSE Framework Implementation. The MUSE framework is implemented in C++. So far, we have implemented a universal wrapper for touch-based devices (e.g., various capacitive and resistive touch overlays, Apple Magic Trackpad or multi-touch touchpads) and corresponding virtual touch sensor module, wrapper for MS Kinect depth sensor and its clones, and touch event matching module. The touch sensor wrapper (*Mwtouch*) uses Linux kernel input API and supports various single and multi-touch devices. Raw events received from a sensor are transformed to TUIO 2.0 messages. *Mwkinect* wrapper processes depth data from MS Kinect depth sensor and provides blob tracking on them. To extract blob description data we used OpenCV image processing library. Since the skeleton-related messages of TUIO 2.0 library were not implemented yet, the initial implementation of the user tracking algorithm does not involve skeletonization part. Thus, the ellipse for matching algorithm represented the whole hand-blob instead of palm area only. *Rendering application* processed data received from wrappers and rendered gesture strokes and blobs representing users' hands. For each touch point and blob we randomly generated color. When touch point was matched, its color changed according to corresponding blob.

5.2 Methodology and Evaluation

Table 1 summarizes processing speed of individual parts of the MUSE framework. *Wrappers.* For both wrapper modules we measured the time elapsed from receiving data (touch event/image frame) from the device to the moment when TUIO 2.0 messages were dispatched. The *mwkinect* wrapper is nearly twice as slow as *mwtouch* wrapper. This is caused by many complex operations which are performed to obtain blob description whereas *mwtouch* transforms input raw data into TUIO 2.0 messages only. Although the average processing time for a single frame is about 23 ms, the processing is still realtime. Frame rate of MS Kinect is up to 30 Hz so there is still ≈10 ms reserve for further improving of the user tracking algorithm.

Virtual touch sensor module processing time was measured in two scenarios: *a)* Localhost – all sensors were connected to a single PC, no networking. *b)* Network connection – each sensor was connected to a different PC and these were connected over local area network (LAN). We performed several types of touch events that were spread across two, three and four sensors. Due to very short distance we observed no significant slowdown using network connection (see Table 1) and the rest of the concatenation accuracy benchmarks were done with the single station setup.

Touch point matching was benchmarked from two perspectives – processing time of the whole MUSE framework aggregator layer and matching algorithm processing speed itself. The aggregator layer processed messages sequentially

Table 1. Processing speed of the MUSE framework modules in [ms]

Wrapper modules		Virtual sensor module		Aggregation module	
mwtouch	**mwkinect**	**localhost**	**LAN**	**total**	**matching alg.**
12.5 ± 8.6	22.6 ± 8.4	0.063 ± 0.003	0.066 ± 0.004	0.3 ± 0.1	0.044 ± 0.02

which resulted in higher computational overhead. Further performance improvement will be realized by batch processing in future version.

Concatenation accuracy. We also investigated the influence of time (Δt) and spatial (Δs) threshold values (see 3.1) on concatenating function since these influence the concatenation algorithm accuracy. Proper threshold values depend on physical parameters of sensor overlays (e.g., frame rate, width of inactive bezel area) and have to be obtained by experimental calibration for each installation. To determine the threshold values we conducted an informal experiment based on drawing lines across the bezels. The measured concatenation algorithm accuracies for 9 combinations of both threshold values are shown in Table 2.

For values of $\Delta s < 450\,px$ and $\Delta t < 1\,s$ approx. 80 % of errors were false negatives (non-concatenated gestures) caused by low frame rate of the touch sensors (up to 15 Hz). Other source of errors produced touch events whose career forms an acute angle $< 20°$ with the sensor bezel and the continuous event started out of the Δs range. We experienced approx. 15 % of false positive errors (i.e. two gestures made by different users) when one user has finished the stroke near the bezel while the other has begun the stroke at the adjacent sensor within the range of Δs. Since we benchmarked only the concatenation accuracy, this type of error should disappear with additional touch point matching.

This benchmark pointed out the weak points of the initial version of the virtual sensor algorithm. Thus the future versions of the concatenation algorithm will have to keep tracking not only each contact position, but also the direction vector and the velocity.

Matching Accuracy. Due to using only the depth sensor of MS Kinect (which provides resolution only up to 640×480 pixels) we struggled with inaccurate positions of blobs. This resulted in reduced matching precision and occurrence of approx. 10 % of false negatives for touch points that were close to the edge of an ellipse (approx. 10-pixel diameter).

In future versions we are going to combine data from both depth sensor and VGA camera which will increase the accuracy and will reduce the loss of blob details (e.g., fingers). Further improvement would come with the next generation of depth sensors equipped with higher resolution IR-sensors. The other issue was the occlusion of hands resulting in merging the relevant blobs into a single (and bigger) one. We identified the problem in the blob tracking algorithm which will be one of the focal points of our future research too.

Table 2. Concatenation algorithm accuracy for various Δt and Δs [%]

$\Delta s \backslash \Delta t$	**0.3** s	**0.5** s	**1** s
300 px	65	70	68
375 px	69	67	72
450 px	68	78	89

Summary. Although only the key modules of the framework were implemented so far, our initial observations are quite encouraging. The key parts of the framework – sensor coupling with concatenation algorithm and touch point matching (even the naive implementation) are very fast (approx. 1000× faster than wrapper processing speed). We reached real-time processing speed with enough time and spare resource capacity for future extensions.

6 Conclusion and Future Work

In this paper, we presented techniques for *a)* assembling touch overlay panels into a nearly seamless multi-touch input layer called *virtual sensor* and *b)* tracking users with further association of input events they performed, which consist of *user tracking algorithm* and *touch point matching algorithm* To overcome the bad performance of the existing body tracking algorithm in top-view tabletop setup, we designed the *relaxed user tracking algorithm.* We integrated these into the MUSE framework which allows rapid development of interactive input layers for high-resolution visualization systems. Existing commercial solutions focus mainly on bare multi-touch capability. Our approach enables building scalable multi-touch surface with additional user distinguishing and association of input actions with the users. These features are crucial for building user-aware interactive environments for the next generation of group collaborative systems. The evaluation of our prototype implementation confirmed usability of commodity devices in such systems.

We continue in improving algorithms and remaining parts of the framework. Virtual sensor algorithm will be extended to dynamic adjustment of time and spatial threshold variables, which will improve the concatenation precision. Relaxed user tracking will be extended with additional blob-contour recognition using both depth sensor and VGA camera to avoid blob-merging noted in previous section. The missing part of the algorithm using skeletonization is currently being implemented as well. Last but not least, we are going to improve the touch point matching algorithm to exclude matches in unexpected areas, such as in the middle of user's palm or at wrist, where usually no one touches the surface (it could also mean the user touches the surface under the arm of another). Concurrently, we are going to adopt the MUSE framework for several graphic rendering platforms, e.g., SAGE [25].

The principles of the framework and algorithms are not limited to touch-based interaction only, but may serve as complex multi-modal interaction input layer. For example, besides coupling multiple touch sensors, it is possible to couple several depth sensor to increase provided image resolution. The algorithms showed to be promising for further exploration and they open a wide range of possible directions for future work, some of which we only touched upon in the paper.

Acknowledgments. This work has been kindly supported by CESNET Large Infrastructure Project (LM2010005).

References

1. Leigh, J.: Cyber-commons: merging real and virtual worlds. Communications of the ACM 51(1), 82–85 (2008)
2. Schmidt, G., et al.: A Survey of Large High-Resolution Display Technologies, Techniques, and Applications. In: Virtual Reality Conference 2006, pp. 223–236 (2006)
3. Teiche, A., et al.: Multitouch Technologies. NUI Group (2009), http://nuicode.com/projects/wiki-book/files
4. Stødle, D.: Device-Free Interaction and Cross-Platform Pixel Based Output to Display Walls. Ph.d. thesis, Uni. of Tromsø (2009)
5. Dietz, P., et al.: DiamondTouch: A Multi-User Touch Technology. In: User Interface Software and Technology 2001, pp. 219–226 (2001)
6. Schmidt, D., et al.: HandsDown: Hand-contour-based user identification for interactive surfaces. In: Nordic Human-Computer Interaction 2010, pp. 432–441 (2010)
7. Tanase, C.A., et al.: Detecting and Tracking Multiple Users in the Proximity of Interactive Tabletops. Advances in Electrical and Computer Engineering 8(2), 61–63 (2008)
8. Annett, M., et al.: Medusa: A Proximity-Aware Multi-Touch Tabletop. In: User Interface Software and Technology 2011, pp. 337–346 (2011)
9. Richter, S., Holz, C., Baudisch, P.: Bootstrapper: Recognizing Tabletop Users by their Shoes. In: Human Factors in Computing Systems 2012, p. 4 (2012)
10. Wilson, A.D., Hrvoje, B.: Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In: User Interface Software and Technology 2010 (2010)
11. Gjerlufsen, T., et al.: Shared Substance: developing flexible multi-surface applications. In: Human Factors in Computing Systems 2011, pp. 3383–3392 (2011)
12. Ponto, K., et al.: CGLXTouch: A multi-user multi-touch approach for ultra-high-resolution collaborative workspaces. Future Generation Computer Systems 27(6), 649–656 (2010)
13. Jagodic, R., et al.: Enabling multi-user interaction in large high-resolution distributed environments. Future Generation Computer Systems, 914–923 (2010)
14. Renambot, L., et al.: SAGE: the Scalable Adaptive Graphics Environment. In: Workshop on Advanced Collaborative Environments 2004, p. 8 (2004)
15. Kaltenbrunner, M., Bencina, R.: reacTIVision (2012), http://reactivision.sourceforge.net/
16. Kim, M., et al.: Design and Development of a Distributed Tabletop System using EBITA Framework. Digital Media, 1–6 (2009)
17. Tuddenham, P., Robinson, P.: T3: A Toolkit for High-Resolution Tabletop Interfaces. In: Computer Supported Cooperative Work 2006, pp. 3–4 (2006)
18. Serrano, M., et al.: The OpenInterface Framework: a tool for multimodal interaction. In: Human Factors in Computing Systems, pp. 3501–3506 (2008)
19. Figueroa, P., et al.: InTml: a description language for VR applications. In: International Conference on 3D Web Technology 2002, pp. 53–58 (2002)
20. Shotton, J., et al.: Real-time human pose recognition in parts from single depth images. In: Computer Vision and Pattern Recognition 2011, pp. 1297–1304 (2011)
21. Chang, F., Chen, C., Lu, C.: A linear-time component-labeling algorithm using contour tracing technique. Comput. Vis. Image Underst. 93(2), 206–220 (2004)

22. Senior, A., et al.: Appearance models for occlusion handling. Image and Vision Computing 24(11), 1233–1243 (2006)
23. Zhang, T.Y., Suen, C.Y.: A fast parallel algorithm for thinning digital patterns. Communications of the ACM 27(3), 236–239 (1984)
24. Kaltenbrunner, M.: TUIO 2.0 (2011), http://www.tuio.org/?tuio20
25. Leigh, J., et al.: SageCommons (2012), http://www.sagecommons.org/

Author Index