

David Cohen (Ed.)

LNCS 6308

Principles and Practice of Constraint Programming – CP 2010

16th International Conference, CP 2010
St. Andrews, Scotland, September 2010
Proceedings

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

David Cohen (Ed.)

Principles and Practice of Constraint Programming – CP 2010

16th International Conference, CP 2010
St. Andrews, Scotland, September 6-10, 2010
Proceedings

Volume Editor

David Cohen
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey, TW20 0EX
United Kingdom
E-mail: d.cohen@rhul.ac.uk

Library of Congress Control Number: 2010933521

CR Subject Classification (1998): F.4.1, F.2, F.3, G.2, F.1, E.1

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-642-15395-X Springer Berlin Heidelberg New York
ISBN-13 978-3-642-15395-2 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2010
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The 16th annual International Conference on the Principles and Practice of Constraint Programming (CP 2010) was held in St. Andrews, Scotland, during September 6–10, 2010. We would like to thank our sponsors for their generous support of this event.

This conference is concerned with all aspects of computing with constraints, including: theory, algorithms, applications, environments, languages, models and systems.

We received a wide variety of submissions, each of which was reviewed by at least three referees. Referees were chosen for each submission by an initial bidding process where Program Committee members chose papers from their area of interest. The range of expertise represented by the large Program Committee meant that almost all submissions were reviewed by subject experts on the Program Committee, or by colleagues chosen by members of the Program Committee for their particular expertise. Papers were solicited either as long (15 page), or short (8 page) submissions. Short-paper submissions were refereed to exactly the same high standards as long-paper submissions but naturally were expected to contain a smaller quantity of new material. Thus there is no distinction in these proceedings between short and long papers. I used the excellent EasyChair conference management system to support this process of reviewing, and for the collation and organization of these proceedings.

Submissions were made either to the applications track or to the research track. There were 101 (23 short) research track submissions of which 36 (8 short) were accepted, which is a 36% (35% of short) acceptance rate. Applications track submissions received special consideration and the acceptance rate was significantly higher than for the research track. There were 21 (2 short) applications track submissions of which 11 (1 short) were accepted for a 52% (50% of short) acceptance rate.

There were three invited talks from distinguished scientists: Robert Nieuwenhuis, Edward Tsang and Moshe Vardi. These proceedings include abstracts of each of their presentations. Details of the wide variety of workshops and the four tutorials that took place as part of the conference are also included.

I would like to thank the Association for Constraint Programming (ACP) for inviting me to be Program Chair. It has been a rewarding experience, not only because of the high quality of the papers submitted, but also for the help so readily given to me by friends and colleagues around the world.

I want to thank all of the authors for submitting such interesting papers. It is their hard work that makes the conference so interesting, but the high quality of the submissions also makes the decision process challenging. I would also like to thank the members of the Program Committee for agreeing to help in the

first place, and for the excellent standard of reviewing. My particular thanks go to those people who provided fourth reviews when decisions were most difficult.

Several people deserve special mention. Karen Petrie, who, as Conference Chair, saved my life on many occasions. Ian Gent, who, as the Program Chair for CP 2009, was a source of invaluable information, warnings and good advice. Barry O'Sullivan for his speedy replies to questions, and general support. Peter Stuckey, who was the Applications Track Chair, Pedro Meseguer for the workshop organization, Thomas Schiex for managing the tutorials and Peter Nightingale and Standa Zivny for running the doctoral program.

The local Organizing Committee always put in a great deal of work, often unthanked, to make a conference like this successful. Andrea Rendl as Publicity Chair and Neil Moore who ran the website helped me a lot during the build up to CP 2010.

September 2010

David Cohen

Distinguished Papers

The Program Committee chose one paper from the Research Track, one paper from the Applications Track and one Student Paper to be recognized as achieving the highest standard submitted in their category.

Best Research Paper

Testing Expressibility Is Hard, by Ross Willard.

Best Applications Paper

Applying Constraint Programming to Identification and Assignment of Service Professionals, by Sigal Asaf, Haggai Eran, Yossi Richter, Daniel P. Connors, Donna L. Gresh, Julio Ortega and Michael J. Mcinnis.

Best Student Paper

Computing the Density of States of Boolean Formulas, by Stefano Ermon, Carla Gomes and Bart Selman.

Workshops and Tutorials

Workshops

As part of CP 2010 a number of workshops took place the day before the main conference, on September 6, 2010.

- 9th Workshop on Constraint Modelling and Reformulation (ModRef 2010)
- 10th Workshop on Symmetry in Constraint Satisfaction Problems (SymCon 2010)
- 7th Workshop on Local Search Techniques in Constraint Satisfaction (LSCS 2010)
- Third Workshop on Quantification in Constraint Programming (QiCP 2010)
- 10th Workshop on Preferences and Soft Constraints (Soft 2010)
- Third Workshop on Techniques for Implementing Constraint Programming Systems (TRICS 2010)
- First Workshop on Constraint Reasoning and Graphical Structures
- Third Workshop on Constraint Reasoning and Optimization for Computational Sustainability (CROCS at CP-10)

Tutorials

Four tutorial presentations were given during the main program of the conference.

- *Distributed CSP*, by Amnon Meisels.
- *SAT with Many Cores*, by Youssef Hamadi.
- *The Valued CSP (VCSP)*, by Martin Cooper, Peter Jeavons and Simon de Givry.
- *Backdoors in CSPs*, by Barry O’Sullivan.

Organization

Executive Committee

Conference Chair	Karen Petrie (University of Dundee, UK)
Program Chair	David Cohen, (Royal Holloway, University of London, UK)
Applications Chair	Peter Stuckey (University of Melbourne, Australia)
Workshop Chair	Pedro Meseguer (IIIA-CSIC, Spain)
Tutorial Chair	Thomas Schiex (INRA Toulouse, France)
Doctoral Program	Peter Nightingale (University of St. Andrews, UK)
Chairs	Standa Živný (University of Oxford, UK)
Sponsorship Chair	Ian Miguel (St. Andrews, UK)
Publicity Chair	Andrea Rendl (St. Andrews, UK)

Sponsors

4C - Cork Constraint Computational Centre
ACP, Association for Constraint Programming
Google, EMEA University Programs
IBM Research
Institute for Computational Sustainability, Cornell University, USA
NICTA, National Information and Communications Technology, Australia
Optimisation for Sustainable Development, Ecole Polytechnique, France
SICS - Swedish Institute of Computer Science

Program Committee

Roman Bartak	Charles University, Czech Republic
Peter van Beek	University of Waterloo, Canada
Hubie Chen	Universitat Pompeu Fabra, Spain
Andy Chun	City University of Hong Kong, Hong Kong
David Cohen	Royal Holloway, University of London, UK
Martin Cooper	IRIT-UPS, Toulouse, France
Victor Dalmau	Universitat Pompeu Fabra, Spain
Rina Dechter	University of California, Irvine, USA
Alan Frisch	University of York, UK
Ian Gent	University of St. Andrews, UK
Carla Gomes	Cornell University, USA
Emmanuel Hebrard	Cork Constraint Computation Centre, Ireland
Brahim Hnich	Izmir University of Economics, Turkey
Peter Jeavons	University of Oxford, UK
George Katsirelos	CRIL, CNRS, France
Zeynep Kiziltan	University of Bologna, Italy

Jimmy Lee	The Chinese University of Hong Kong, Hong Kong
Pedro Meseguer	IIIA-CSIC, Spain
Laurent Michel	University of Connecticut, USA
Peter Nightingale	University of St. Andrews, UK
Barry O'Sullivan	Cork Constraint Computation Centre, Ireland
Justin Pearson	Uppsala University, Sweden
Karen Petrie	University of Dundee, University of Dundee, UK
Claude-Guy Quimper	Université Laval, France
Emma Rollon	Technical University of Catalonia, Spain
Francesca Rossi	University of Padova, Italy
Thomas Schiex	INRA Toulouse, France
Christian Schulte	KTH - Royal Institute of Technology, Sweden
Meinolf Sellmann	Brown University, USA
Helmut Simonis	Cork Constraint Computation Centre, Ireland
Kostas Stergiou	University of the Aegean, Greece
Peter Stuckey	University of Melbourne, Australia
Mark Wallace	Monash University, Australia
Toby Walsh	NICTA, Australia
Roland Yap	National University of Singapore, Singapore
Standa Živný	University of Oxford, UK

Organizing Committee

David Cohen	Royal Holloway, University of London
Maria Fox	University of Strathclyde
Alan Frisch	University of York
Ian Gent	University of St. Andrews
Youssef Hamadi	Microsoft Research, Cambridge
Peter Jeavons	University of Oxford
Chris Jefferson	University of St. Andrews
Tom Kelsey	University of St. Andrews
Andrei Krokhin	University of Durham
Steve Linton	University of St. Andrews
Derek Long	University of Strathclyde
Ian Miguel	University of St. Andrews
Karen Petrie	University of Dundee
Patrick Prosser	University of Glasgow
Barbara Smith	University of Leeds
Edward Tsang	University of Essex

Additional Reviewers

Kiyam Ahmadizadeh
 Ozgur Akgun
 Alexandre Albore
 Carlos Ansétegui
 Magnus Ågren
 Thanasis Balafoutis
 Mauro Bampo
 Ralph Becket
 Christian Bessiere
 Manuel Bodirsky
 Andrej Bogdanov
 Simone Bova
 Martin Brain
 Sebastian Brand
 Ken Brown
 Andrei Bulatov
 Hadrien Cambazard
 Catarina Carvalho
 Kenil Cheng
 Geoffrey Chu
 Páidí Creed
 James Cussens
 Jessica Davies
 Simon de Givry
 Thibaut Feydy
 Pierre Flener
 Yong Gao
 Marco Gavanelli
 Vibhav Gogate
 Laurent Granvilliers
 Magnus gren
 Tarik Hadzic
 Daniel Harabor
 Martin Henz
 Tim Januschowski
 Christopher Jefferson
 Serdar Kadioglu
 Kalev Kask
 Richard Kelly
 Lars Kotthoff
 Lukas Kroc
 Andrei Krokhin
 Daniel Kudenko
 Mikael Zayenz Lagerkvist

Arnaud Lallouet
 Javier Larrosa
 Yat Chiu Law
 Ronan LeBras
 Chavalit Likitvivanavong
 Michele Lombardi
 Derek Long
 Ines Lynce
 Michael Lyu
 Yuri Malitsky
 Toni Mancini
 Radu Marinescu
 Joao Marques-Silva
 Robert Mateescu
 Jacopo Mauro
 Chris Mears
 Deepak Mehta
 Ian Miguel
 Michela Milano
 Neil Moore
 Nina Narodytska
 Gustav Nordh
 Olga Ohrimenko
 Albert Oliveras
 Lars Otten
 Justyna Petke
 Maria Silvia Pini
 Cédric Pralet
 Steve Prestwich
 Steven Prestwich
 Luis Quesada
 Miguel Ramírez
 Andrea Rendl
 Enric Rodríguez-Carbonell
 Roberto Rossi
 Michel Rueher
 Tyrel Russell
 Ashish Sabharwal
 Domenico Salvagnin
 Horst Samulowitz
 Andrew Santosa
 Tom Schrijvers
 Brand Sebastian
 Charles F.K. Siu

XIV Organization

David Stynes
Pramudi Suraweera
Guido Tack
Michael Thomas
Evgenij Thorstensen
Marc Thurley
Gilles Trombettoni
Jeremie Vautard

Gérard Verfaillie
Magnus Wahlström
Richard Wallace
May H.C. Woo
Michal Wrona
Yuanlin Zhang
Roie Zivan

Association for Constraint Programming

The Association for Constraint Programming (ACP) aims at promoting constraint programming in every aspect of the scientific world, by encouraging its theoretical and practical developments, its teaching in academic institutions, its adoption in the industrial world, and its use in applications.

The ACP is a non-profit association, which uses the profit of the organized events to support future events or activities. At any given time members of the ACP are all attendees CP conferences of the past five years, and all members of the Program Committee of the current CP conference.

The ACP is led by an Executive Committee (EC), which takes all the decisions necessary to achieve the goals of the association. In particular, the ACP EC organizes an annual international conference on constraint programming: selecting the venue and choosing the Program and Conference Chairs. This annual conference includes a doctoral program, which is intended to encourage doctoral students to work on CP and to participate in the CP conference.

The ACP EC maintains a website (<http://www.4c.ucc.ie/a4cp/>) about all aspects of CP, and publishes a quarterly newsletter about CP events.

ACP Executive Committee

President: Barry O'Sullivan

Secretary: Jimmy H.M. Lee

Treasurer: Thomas Schiex

Conference Coordinator: Pedro Meseguer

Others:

- John Hooker
- Karen Petrie
- Peter Stuckey
- Roland Yap

Table of Contents

Invited Talks

SAT Modulo Theories: Getting the Best of SAT and Global Constraint Filtering	1
<i>Robert Nieuwenhuis</i>	
Constraint-Directed Search in Computational Finance and Economics	3
<i>Edward Tsang</i>	
Constraints, Graphs, Algebra, Logic, and Complexity	8
<i>Moshe Y. Vardi</i>	

Distinguished Papers

Testing Expressibility Is Hard	9
<i>Ross Willard</i>	
Applying Constraint Programming to Identification and Assignment of Service Professionals	24
<i>Sigal Asaf, Haggai Eran, Yossi Richter, Daniel P. Connors, Donna L. Gresh, Julio Ortega, and Michael J. Mcinnis</i>	
Computing the Density of States of Boolean Formulas	38
<i>Stefano Ermon, Carla P. Gomes, and Bart Selman</i>	

Research Track

Towards Parallel Non Serial Dynamic Programming for Solving Hard Weighted CSP	53
<i>David Allouche, Simon de Givry, and Thomas Schiex</i>	
Making Adaptive an Interval Constraint Propagation Algorithm Exploiting Monotonicity	61
<i>Ignacio Araya, Gilles Trombettoni, and Bertrand Neveu</i>	
Improving the Performance of maxRPC	69
<i>Thanasis Balafoutis, Anastasia Paparrizou, Kostas Stergiou, and Toby Walsh</i>	
Checking-Up on Branch-and-Check	84
<i>J. Christopher Beck</i>	

Spatial, Temporal, and Hybrid Decompositions for Large-Scale Vehicle Routing with Time Windows	99
<i>Russell Bent and Pascal Van Hentenryck</i>	
Decomposition of the NVALUE Constraint	114
<i>Christian Bessiere, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh</i>	
Propagating the Bin Packing Constraint Using Linear Programming	129
<i>Hadrien Cambazard and Barry O’Sullivan</i>	
Sweeping with Continuous Domains	137
<i>Gilles Chabert and Nicolas Beldiceanu</i>	
A New Hybrid Tractable Class of Soft Constraint Problems	152
<i>Martin C. Cooper and Stanislav Živný</i>	
A Propagator for Maximum Weight String Alignment with Arbitrary Pairwise Dependencies	167
<i>Alessandro Dal Palù, Mathias Möhl, and Sebastian Will</i>	
Using Learnt Clauses in MAXSAT	176
<i>Jessica Davies, Jeremy Cho, and Fahiem Bacchus</i>	
Domain Consistency with Forbidden Values	191
<i>Yves Deville and Pascal Van Hentenryck</i>	
Generating Special-Purpose Stateless Propagators for Arbitrary Constraints	206
<i>Ian P. Gent, Chris Jefferson, Ian Miguel, and Peter Nightingale</i>	
Including Ordinary Differential Equations Based Constraints in the Standard CP Framework	221
<i>Alexandre Goldsztejn, Olivier Mullier, Damien Eveillard, and Hiroshi Hosobe</i>	
Structural Tractability of Enumerating CSP Solutions	236
<i>Gianluigi Greco and Francesco Scarcello</i>	
Diversification and Intensification in Parallel SAT Solving	252
<i>Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Sais</i>	
A Systematic Approach to MDD-Based Constraint Programming	266
<i>Samid Hoda, Willem-Jan van Hoeve, and J.N. Hooker</i>	
A Complete Multi-valued SAT Solver	281
<i>Siddhartha Jain, Eoin O’Mahony, and Meinolf Sellmann</i>	
Exact Cover via Satisfiability: An Empirical Study	297
<i>Tommi Junttila and Petteri Kaski</i>	

On the Complexity and Completeness of Static Constraints for Breaking Row and Column Symmetry	305
<i>George Katsirelos, Nina Narodytska, and Toby Walsh</i>	
Ensemble Classification for Constraint Solver Configuration	321
<i>Lars Kotthoff, Ian Miguel, and Peter Nightingale</i>	
On Testing Constraint Programs	330
<i>Nadjib Lazaar, Arnaud Gotlieb, and Yahia Lebbah</i>	
On the Containment of Forbidden Patterns Problems	345
<i>Florent Madelaine</i>	
Improving the Floating Point Addition and Subtraction Constraints	360
<i>Bruno Marre and Claude Michel</i>	
The Lattice Structure of Sets of Surjective Hyper-Operations	368
<i>Barnaby Martin</i>	
Constraint Based Scheduling to Deal with Uncertain Durations and Self-Timed Execution	383
<i>Michele Lombardi and Michela Milano</i>	
Local Consistency and SAT-Solvers	398
<i>Justyna Petke and Peter Jeavons</i>	
Heuristics for Planning with SAT	414
<i>Jussi Rintanen</i>	
Value-Ordering Heuristics: Search Performance vs. Solution Diversity . . .	429
<i>Yevgeny Schreiber</i>	
A New $\mathcal{O}(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints	445
<i>Andreas Schutt and Armin Wolf</i>	
A Generic Visualization Platform for CP	460
<i>Helmut Simonis, Paul Davern, Jacob Feldman, Deepak Mehta, Luis Quesada, and Mats Carlsson</i>	
Database Constraints and Homomorphism Dualities	475
<i>Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan</i>	
A Box-Consistency Contractor Based on Extremal Functions	491
<i>Gilles Trombettoni, Yves Papegay, Gilles Chabert, and Odile Pourtallier</i>	
Exponential Propagation for Set Variables	499
<i>Justin Yip and Pascal Van Hentenryck</i>	

Applications Track

An Empirical Study of Optimization for Maximizing Diffusion in Networks	514
<i>Kiyan Ahmadizadeh, Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal</i>	
An Integrated Modelling, Debugging, and Visualisation Environment for G12	522
<i>Andreas Bauer, Viorica Botea, Mark Brown, Matt Gray, Daniel Harabor, and John Slaney</i>	
Global Constraints on Feature Models	537
<i>Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Dođru</i>	
Constraint Programming for Mining n-ary Patterns	552
<i>Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux</i>	
An Integrated Business Rules and Constraints Approach to Data Centre Capacity Management	568
<i>Roman van der Krogt, Jacob Feldman, James Little, and David Stynes</i>	
Context-Sensitive Call Control Using Constraints and Rules	583
<i>David Lesaint, Deepak Mehta, Barry O’Sullivan, Luis Quesada, and Nic Wilson</i>	
Load Balancing and Almost Symmetries for RAMBO Quorum Hosting	598
<i>Laurent Michel, Alexander A. Shvartsman, Elaine Sonderegger, and Pascal Van Hentenryck</i>	
Testing Continuous Double Auctions with a Constraint-Based Oracle . . .	613
<i>Roberto Castañeda Lozano, Christian Schulte, and Lars Wahlberg</i>	
A Safe and Flexible CP-Based Approach for Velocity Tuning Problems	628
<i>Michaël Soullignac, Michel Rueher, and Patrick Taillibert</i>	
Contingency Plans for Air Traffic Management	643
<i>Karl Sundequist Blomdahl, Pierre Flener, and Justin Pearson</i>	
Author Index	659

SAT Modulo Theories: Getting the Best of SAT and Global Constraint Filtering

Robert Nieuwenhuis*

Technical Univ. of Catalonia (UPC), Barcelona, Spain

The propositional satisfiability problem (SAT) is one of the simplest instances of Constraint Programming (CP): variables are bi-valued (can only take values 0 or 1), and all constraints are *clauses* (disjunctions of literals) like $x \vee \bar{y} \vee \bar{z}$ (meaning that $x = 1$ or $y = 0$ or $z = 0$).

In spite of its simplicity, SAT has become very important for practical applications, especially in the multi-billion industry of electronic design automation (EDA), and, in general, hardware and software verification. Research on SAT has been pushed by these huge industrial needs and resources, in a very pragmatic way: prestigious conferences are eager to publish papers describing how to improve performance on their real-world problems, even if these improvements are not based on highly original techniques (in contrast with conferences like CP, which tend to prefer new ideas, even if they are tested only on *academic* random or artificial problem instances).

As a result, modern SAT solvers work impressively well on real-world problems from *many* sources, using a *single, fully automatic, push-button* strategy. Hence, modeling and using SAT is essentially a *declarative* task. On the negative side, propositional logic is a very low level language and hence modeling and encoding *tools* are required, and also optimization aspects are not that well studied.

Sophisticated encodings into SAT have been developed for many constraints that are typical in EDA and verification applications, such as arrays, congruences, or Difference Logic and other fragments of linear arithmetic. However, in many cases such encodings become too large, and/or behave poorly. *SAT Modulo Theories* (SMT) was developed as an answer to this situation. The idea is to encode only part of the constraints into SAT. The remaining constraints are considered as a background “theory”. Similarly to the filtering algorithms in Constraint Programming, the *Theory Solver* uses efficient *specialized algorithms* to detect additional propagations and inconsistencies with respect to this theory.

For example, given a large input with clauses such as $3x + 4y \leq 6 \vee \bar{z} \vee \dots$ the SAT component of the SMT solver will consider the linear arithmetic literals like $3x + 4y \leq 6$ as any other (meaningless) propositional literal, but in addition there is a Theory Solver using a simplex algorithm to check whether the current set (conjunction) of linear arithmetic literals is *T-consistent*, or whether it *T-propagates* some other arithmetic literal occurring in the clause set.

What distinguishes SMT from complete CP search techniques with global constraint filtering algorithms is that SMT maintains SAT’s extremely successful

* Partially supported by Spanish Min. of Science & Innovation, LogicTools-2 project (TIN2007-68093-C02-01).

tightly interconnected ingredients, such as *clause learning*, *clause forgetting*, *conflict analysis*, *backjumping*, and *activity-based variable selection heuristics*.

In this talk we first give an overview of SMT, the *DPLL(T)* approach to SMT [NOT06], and its implementation in our Barcelegic SMT tool. Then we discuss current work on the development of SMT technology for hard combinatorial (optimization) problems outside the usual verification applications. The aim is to obtain the best of several worlds, combining the advantages inherited from SAT: efficiency, robustness and automation (no need for *tuning*) and CP features such as rich modeling languages, special-purpose filtering algorithms (for, e.g., planning, scheduling or timetabling constraints), and sophisticated optimization techniques. We give several examples and discuss the impact of aspects such as first-fail heuristics vs activity-based ones, realistic structured problems vs random or handcrafted ones, and lemma learning.

Further reading: The recent Handbook of Satisfiability [BHvMW09] has chapters on all the main aspects of SAT, from underlying theoretical results to implementation techniques and applications, with many further references. In particular, it contains a very nice chapter on SMT [BSST09].

References

- [BHvMW09] Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (February 2009)
- [BSST09] Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability Modulo Theories. In: Biere et al. [BHvMW09], ch. 26, vol. 185, pp. 825–885 (February 2009)
- [NOT06] Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). Journal of the ACM 53(6), 937–977 (2006)

Constraint-Directed Search in Computational Finance and Economics

Edward Tsang

Centre for Computational Intelligence in Finance & Economics (CCFEA)
University of Essex, Colchester, UK
edward@essex.ac.uk

1 Use the Force

Constraints shield solutions from a problem solver. However, in the hands of trained constraint problem solvers, the same constraints that create the problems in the first place can also guide problem solvers to solutions. Constraint satisfaction is all about learning how to flow with the force of the constraints.

Examples of using constraints to guide one's search are abundant in complete search methods (e.g. see [12]). Lookahead algorithms propagate constraints in order to (a) reduce the remaining problem to smaller problems and (b) detect dead-ends. Dependency-directed backtracking algorithms use constraints to identify potential culprits in dead-ends. This helps the search to avoid examining (in vain) combinations of variables assignments that do not matter.

Constraint-directed search is used in stochastic search too. Constraints were used in Guided Local Search (GLS) [3] and Guided Genetic Algorithm (GGA) [4] to guide the search to promising areas of the search space. In stochastic methods, a constraint satisfaction problem is handled as an optimization problem, where the goal is to minimize the number of constraints violated. The approach in GLS is to use constraints to augment the objective function. This helps local search to escape local optima. GGA uses the GLS penalty scheme to change the behaviour of genetic algorithms. This results in a more robust algorithm which finds quality results consistently. GLS and GGA have been applied to many optimization problems, including the well-known travelling salesman problem and the quadric assignment problem.

The GLS idea was generalized to “penalties” and “incentives” in evolutionary computation. This paper explains how such ideas were applied to two applications in finance and economics: financial forecasting and automated bargaining.

2 Constraints in Financial Forecasting

In forecasting, the goal is to predict the value of a variable, which defines the target. The challenge in forecasting is (i) to find a set of variables, and (ii) to find a function that maps these variables to the target. There is no limit in the format of this function. It can be a mathematical function. It can also be a program procedure.

There is no guarantee that such functions exist. If they do, then finding the relevant variables is essential for their discovery. EDDIE (which stands for Evolutionary Dynamic Data Investment Evaluator) is a framework for finding such functions [5]. Experts channel their financial expertise into the system through the suggestion of variables. EDDIE attempts to find functions that map these variables to the target.

EDDIE attempts to predict a particular form of patterns: whether prices will go up by $r\%$ or more within the next n days. (Here r could be a negative number). In that case, the target can be represented by a Boolean variable T . If T is true, it means prices will go up by $r\%$ or more within the next n days, which represents an investment opportunity. For example, domain experts may suggest that the current price, the “50 days moving average” and “volatility” (which could be measured by the normalized standard deviation of the previous, say, 25 days) are indicators of the future price. An example of a function is:

```
IF the current price is 6.24% above the 50 days moving average
  THEN IF volatility is above 1.85,
    THEN T is True;
    ELSE T is False;
ELSE IF the current price is 12.49% below the 50 days moving average
  THEN T is False;
  ELSE T is True;
```

In this example, the function is represented by a tree. EDDIE is responsible for finding the structure of the tree, as well as the thresholds such as 6.24%, 1.85.

The search for variables is crucial to the success of forecasting. This is the job of the finance experts, which will not be discussed here. (This job can be helped by EDDIE, see [6]). Faced by EDDIE is a huge search space of tree structures and thresholds. EDDIE searches the space with genetic programming. Pretty standard genetic programming techniques were adopted, except the use of constraints, which is described below.

In EDDIE, precision refers to the percentage of “True” predictions that turn out to be correct in reality. Recall refers to the percentage of investment opportunities that were correctly predicted “True” by EDDIE. Failure in picking an opportunity is not as serious as making a wrong decision to invest, because the latter could lead to losses. That means precision is more important than recall in financial forecasting. Having said that, if a forecasting tool fails to pick up any investment opportunities at all (i.e. recall=0), then this tool is useless. Therefore, one would like to have a handle to balance between precision and recall. This is attempted by FGP2, a version of EDDIE. Following is a brief summary of FGP2; details can be found in [7].

FGP2 aims to concentrate the search on areas of the space where trees have higher precision. To achieve that, FGP2 augmented the objective function with a constraint. The augmented objective function encouraged trees that make a certain percentage of their predictions “True”. If the percentage of “True” predictions by a tree is not within a range constrained by the user, its fitness is

significantly reduced. This range constraint is set by the user; it typically reflects conditions of the current market.

Trees capture patterns in the data. The EDDIE experience was that, even with variables drawn from text-books (namely technical trading indicators), patterns could be found in some of the stocks [7]. With variables of better quality, patterns with extremely high precision could be found [8]. Patterns do not appear in all stocks. Besides, the market has been changing very fast in recent years (with the significant growth of algorithmic trading), which hinders learning. Nevertheless, one does not have to find all patterns to benefit from forecasting. A single opportunity, if detected, could provide a trader with valuable reward. Whenever patterns exist, having a forecasting tool like EDDIE is better than not.

3 Constraints in Automated Bargaining

Game theory is often used in a political or military context to explain conflicts between countries. More recently it has been used to map trends in the business world, ranging from how cartels set prices to how companies can better sell their goods and services in new markets. It has become an important area in economics, for which Nobel Prizes have been awarded (e.g. Aumann and Schelling in 2005).

Bargaining is a main subject in game theory. One of the fundamental bargaining models was Rubinstein's 1982 model. Under this model, two players bargain to share a pie. They make alternative offers. For example, the first player may offer to take 65% of the pie. The second player may either accept it or reject the 35% offered. If he rejects this offer, he will have to give a counter offer, e.g. he may ask for 51%. However, both players' utilities drop exponentially over time. That means getting 51% in the second round may not worth as much as accepting 35% in the first round. This motivates both players to accept an offer as soon as possible. It is worth pointing out that the two players may have different utility discount rates. The value of the discount rate determines their bargaining power. A player with a higher discount rate is in a weaker position to bargain.

A player's optimal strategy depends on the other player's strategy. Subgame equilibrium is the optimal strategy by both players, given their belief of their opponent's strategies. To derive the subgame equilibrium, Rubinstein assumed complete information, i.e. each player knows both discount rates, and knows that the opponent knows. Rubinstein also assumed perfect rationality by both players [9]. Subgame equilibrium was derived recursively by Rubinstein: to calculate the first player's optimal strategy, one has to solve the subproblem of the second player's strategy. This in turn can be calculated by the first player's optimal strategy in the third round should the second player make a counter offer in the previous round. The subproblems can be solved recursively till both players' utilities drop to a fix point.

In game theory, subgame equilibrium is typically derived mathematically. There are two serious drawbacks in this approach. Firstly, it assumes perfect rationality in decision making. In practice, decision making often involves

computation (chess is a good example). Therefore, computational intelligence determines the effective rationality (I call this the CIDER theory, see [10]). Secondly, mathematical derivation of subgame equilibrium is laborious. A slight change of the bargaining model (for example, when a player has an outside option which guarantees him, say, 36% of the pie) would typically require complete revision of the derivation.

The above drawbacks of the mathematical approach motivate a co-evolutionary approach, where each of the two players is modelled by a population of strategies [11]. A strategy's fitness is evaluated through playing it with strategies by the opponent. In evolutionary computation, a strategy's chance of survival depends on its fitness. That means under this approach, the perfect rationality assumption is replaced by reinforcement learning, which is closer to reality. Besides, this approach is robust: it can easily cope with slight changes to the bargaining model. It can easily capture asymmetric information or asymmetric ability by the two players.

Jin et al used genetic programming to approximate subgame equilibrium [11]. Bidding strategies were represented by functions. Under this approach, each player searches in the space of functions. Unfortunately, the search space is huge. Besides, only a very small proportion of the functions in the search space are sensible. For example, a random strategy would typically return a bid of below 0% or above 100% of the pie. Standard genetic programming failed to find sensible strategies consistently.

Following EDDIE's experience, Jin and Tsang used constraints to guide the search. To do so, desirable attributes were identified for bidding strategies. Firstly, a strategy should return a value between 0 and 1. Secondly, the value that a bidding strategy returns should ideally be inversely proportional to the player's own utility discount rate. Thirdly, the value that a strategy returns should ideally be proportional to the opponent's discount rate. These desirable attributes were translated into incentives, which augmented the objective function.

With the help of incentives, the majority of the populations contained usable bidding strategies (which demand a value between 0% and 100%). The subgame equilibrium found by co-evolution was very close to the theoretical solutions in Rubinstein's 1982 bargaining model. With minor modifications, the programs were applicable to variations of Rubinstein's bargaining model [9]. For these simple bargaining models, the subgame equilibrium found by co-evolution approximated the theoretical solutions. These results suggest that constraint-directed co-evolutionary is a useful approach to approximate subgame equilibrium in bargaining.

References

1. Tsang, E.P.K.: Foundations of constraint satisfaction. Academic Press, London (1993)
2. Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier, Amsterdam (2006)

3. Voudouris, C., Tsang, E.P.K.: Guided local search. In: Glover, F. (ed.) *Handbook of Metaheuristics*, pp. 185–218. Kluwer, Dordrecht (2003)
4. Lau, T.L., Tsang, E.P.K.: Guided genetic algorithm and its application to radio link frequency assignment problems. *Constraints* 6(4), 373–398 (2001)
5. Tsang, E.P.K., Yung, P., Li, J.: EDDIE-Automation, a decision support tool for financial forecasting. *Journal of Decision Support Systems, Special Issue on Data Mining for Financial Decision Making* 37(4), 559–565 (2004)
6. Kampouridis, M., Tsang, E.: EDDIE for Investment Opportunities Forecasting: Extending the Search Space of the GP. In: *Proceedings of the IEEE Congress on Evolutionary Computation 2010, Barcelona, Spain (2010)* (to appear)
7. Li, J., Tsang, E.P.K.: Investment decision making using FGP: a case study. In: *Proceedings of Congress on Evolutionary Computation, Washington, DC, USA*, pp. 1253–1259 (1999)
8. Tsang, E.P.K., Markose, S., Er, H.: Chance discovery in stock index option and future arbitrage. *New Mathematics and Natural Computation* 1(3), 435–447 (2005)
9. Rubinstein, A.: Perfect Equilibrium in a Bargaining Model. *Econometrica* 50, 97–110 (1982)
10. Tsang, E.P.K.: Computational intelligence determines effective rationality. *International Journal on Automation and Control* 5(1), 63–66 (2008)
11. Jin, N., Tsang, E.P.K., Li, J.: A constraint-guided method with evolutionary algorithms for economic problems. *Applied Soft Computing* 9(3), 924–935 (2009)
12. Tsang, E.P.K.: Forecasting — where computational intelligence meets the stock market. *Frontiers of Computer Science in China*, pp. 53–63. Springer, Heidelberg (2009)

Constraints, Graphs, Algebra, Logic, and Complexity*

Moshe Y. Vardi

Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.

`vardi@cs.rice.edu`

<http://www.cs.rice.edu/~vardi>

A large class of problems in AI and other areas of computer science can be viewed as constraint-satisfaction problems. This includes problems in database query optimization, machine vision, belief maintenance, scheduling, temporal reasoning, type reconstruction, graph theory, and satisfiability. All of these problems can be recast as questions regarding the existence of homomorphisms between two directed graphs. It is well-known that the constraint-satisfaction problem is NP-complete. This motivated an extensive research program to identify tractable cases of constraint satisfaction.

This research proceeds along two major lines. The first line of research focuses on non-uniform constraint satisfaction, where the target graph is fixed. The goal is to identify those target graphs that give rise to a tractable constraint-satisfaction problem. The second line of research focuses on identifying large classes of source graphs for which constraint-satisfaction is tractable. We show in this talk how tools from graph theory, universal algebra, logic, and complexity theory, shed light on the tractability of constraint satisfaction.

Reference

1. Kolaitis, P.G., Vardi, M.Y.: A logical approach to constraint satisfaction. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) *Complexity of Constraints*. LNCS, vol. 5250, pp. 125–155. Springer, Heidelberg (2008)

* Work supported in part by NSF grants CCR-0311326, CCF-0613889, ANI-0216467, and CCF-0728882.

Testing Expressibility Is Hard

Ross Willard*

Pure Mathematics Department

University of Waterloo

Waterloo, Ontario N2L 3G1 Canada

<http://www.math.uwaterloo.ca/~rdwillar>

Abstract. We study the *expressibility problem*: given a finite constraint language Γ on a finite domain and another relation R , can Γ express R ? We prove, by an explicit family of examples, that the standard witnesses to expressibility and inexpressibility (gadgets/formulas/conjunctive queries and polymorphisms respectively) may be required to be exponentially larger than the instances. We also show that the full expressibility problem is co-NEXPTIME-hard. Our proofs hinge on a novel interpretation of a tiling problem into the expressibility problem.

Keywords: constraint, relation, expressive power, inverse satisfiability, structure identification, conjunctive query, primitive positive formula, polymorphism, domino system, nondeterministic exponential time.

1 Introduction

Given a fixed set Γ of *basic* constraint relations for building constraint programs or satisfaction problems, there are typically other (perhaps useful) *implicit* relations which may be treated as if they were actually present in Γ , without affecting the expressiveness or complexity of Γ .

For example, consider the toy constraint language $\Gamma = \{\rightarrow, U\}$ on the domain $D = \{0, 1, 2, 3, 4, 5\}$, where \rightarrow is the binary relation pictured in Figure 1 and U is the unary relation $\{0, 3\}$.

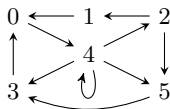


Fig. 1. The binary relation \rightarrow

The unary relation $V = \{3, 4, 5\}$ is an example of an implicit relation of $\{\rightarrow, U\}$. Indeed, whenever we wish to constrain a variable x to V , we can accomplish this by adding three new auxiliary variables a_x, b_x, c_x and imposing

* The support of the Natural Sciences and Engineering Research Council of Canada and the American Institute of Mathematics is gratefully acknowledged.

the basic constraints $a_x \rightarrow b_x$, $b_x \rightarrow x$, $x \rightarrow c_x$, $U(a_x)$, and $U(c_x)$. We say that Γ can *express* V . We might similarly ask: can Γ express the complement of V , i.e., the unary relation $W = \{0, 1, 2\}$? What about the complement of \rightarrow ?

These questions are instances of the *expressibility problem* (also known as the *existential inverse satisfiability problem* [7,6] and the *pp-definability problem* [4]). It is a *structure identification problem* in the sense of [8]. Its answers define what is called the *expressive power* of a constraint language [15], now a key tool in the quest to classify which constraint languages are tractable (e.g., [12,3]).

In this paper we give constructions which show that the general expressibility problem is impossibly hard according to three natural measures.

We thank Matt Valeriote and Moshe Vardi for some helpful discussions.

2 Definitions, Basic Facts, and Statement of Results

Fix a constraint language Γ on a finite domain D . Given an instance $\mathcal{P} = (X, D, \mathcal{C})$ of $\text{CSP}(\Gamma)$, we shall use $\text{Sol}(\mathcal{P})$ to denote the set of all solutions to \mathcal{P} , construed as functions $X \rightarrow D$. If $\mathbf{s} = (s_1, \dots, s_k)$ is a k -tuple of variables from X , then we shall use $\pi_{\mathbf{s}}(\text{Sol}(\mathcal{P}))$ to denote the restriction of $\text{Sol}(\mathcal{P})$ to \mathbf{s} ; i.e.,

$$\pi_{\mathbf{s}}(\text{Sol}(\mathcal{P})) = \{(f(s_1), \dots, f(s_k)) : f \in \text{Sol}(\mathcal{P})\} \subseteq D^k.$$

Definition 1 ([15,5,13]). *Given a constraint language Γ and a k -ary relation R on a domain D , we say that Γ expresses (or generates) R if there exists an instance $\mathcal{P} = (X, D, \mathcal{C})$ of $\text{CSP}(\Gamma)$ and a k -tuple $\mathbf{s} = (s_1, \dots, s_k)$ of variables with $\pi_{\mathbf{s}}(\text{Sol}(\mathcal{P})) = R$. The pair $(\mathcal{P}, \mathbf{s})$ is a witness to the expressibility of R by Γ .*

Cohen and Jeavons [5] have called \mathcal{P} a *gadget* and \mathbf{s} a *construction site* in this context. A witness $(\mathcal{P}, \mathbf{s})$ can be trivially re-formulated as a *conjunctive query* over Γ (in database theory) or as a *primitive positive formula* over Γ (in logic); the latter is an expression of the form $\exists y_1 \dots \exists y_n [C_1 \ \& \ C_2 \ \& \ \dots \ \& \ C_r]$, asserting the existence of auxiliary variables satisfying (with \mathbf{s}) the constraints of \mathcal{P} .

Example 1. In the example from Section 1, let \mathcal{P} be the instance of $\text{CSP}(\Gamma)$ having variable set $\{a, b, c, x\}$ and constraints $((a, b), \rightarrow)$, $((b, x), \rightarrow)$, $((x, c), \rightarrow)$, (a, U) , and (c, U) . \mathcal{P} has exactly four solutions; identifying each solution f_i with its 4-tuple of values $(f_i(a), f_i(b), f_i(x), f_i(c))$, we have

$$\text{Sol}(\mathcal{P}) = \{(0, 4, 3, 0), (0, 4, 4, 3), (0, 4, 5, 3), (3, 0, 4, 3)\}.$$

As the projection of $\text{Sol}(\mathcal{P})$ on its third coordinate (i.e., at x) is $\{3, 4, 5\} = V$, (\mathcal{P}, x) witnesses the fact that Γ can express V . An equivalent primitive positive formula witnessing this is $\exists a \exists b \exists c [a \rightarrow b \ \& \ b \rightarrow x \ \& \ x \rightarrow c \ \& \ U(a) \ \& \ U(c)]$.

Definition 2. *Suppose D is a finite domain, Γ is a constraint language over D , and n, k are positive integers. Let $\mathbf{s} = (s_1, \dots, s_k)$ be a k -tuple of elements from D^n , R a k -ary relation on D , and $h : D^n \rightarrow D$.*

1. $\text{proj}(\mathbf{s}) = \{(\mathbf{s}_1[i], \dots, \mathbf{s}_k[i]) : 1 \leq i \leq n\}$. (Thus $\text{proj}(\mathbf{s}) \subseteq D^k$.)
2. h preserves R at s if $\text{proj}(\mathbf{s}) \not\subseteq R$ or $(h(\mathbf{s}_1), \dots, h(\mathbf{s}_k)) \in R$.
3. h preserves R if h preserves R at every k -tuple in $(D^n)^k$.
4. h is a polymorphism of Γ (of arity n) if h preserves every relation in Γ .

One can show that, for every $n \geq 1$, there exists an instance of $\text{CSP}(\Gamma)$ with variable set D^n whose solutions are precisely the n -ary polymorphisms of Γ . Following Jeavons, Cohen and Gyssens [14,11,15,5], we call this CSP instance the *indicator problem for Γ of order n* and denote it by $\mathcal{J}_n(\Gamma)$.

It is well-known that the polymorphisms of Γ (i) include the projections and (ii) preserve all relations expressed by Γ (see e.g. [15, Lemma 2.18]). From this one can deduce the following connection between expressible relations, polymorphisms, and indicator problems.

Proposition 1. *For any $n, k \geq 1$ and $\mathbf{s} \in (D^n)^k$, the relation S expressed by $(\mathcal{J}_n(\Gamma), \mathbf{s})$ (i) contains $\text{proj}(\mathbf{s})$, and (ii) is contained in every k -ary relation expressible from Γ which contains $\text{proj}(\mathbf{s})$. I.e., S is the smallest k -ary relation expressible from Γ containing $\text{proj}(\mathbf{s})$.*

Note that if R is k -ary and there exists an n -ary polymorphism h of Γ which does not preserve R at some $\mathbf{s} \in (D^n)^k$, then R is not expressible from Γ . When this happens we say that h is a *witness* to the inexpressibility of R from Γ .

Example 2. Returning to the example in Section 1, the 1-ary map $h : D \rightarrow D$ when sends $1 \mapsto 3$, $2 \mapsto 4$, and fixes all other elements of D , is a polymorphism of $\Gamma = \{\rightarrow, U\}$. As $1 \in W = \{0, 1, 2\}$ but $h(1) \notin W$, h does not preserve W at 1; hence W is not expressible from Γ , and h is a witness.

For any k -ary relation R on D , if n is the number of rows of R (i.e., $n = |R|$), then one can construct $\mathbf{s}^{(R)} = (\mathbf{s}_1, \dots, \mathbf{s}_k) \in (D^n)^k$ so that $\text{proj}(\mathbf{s}^{(R)}) = R$. As R is expressible from Γ exactly when the smallest k -ary relation expressible from Γ and containing R is R itself, it follows from Proposition 1 that *either* $(\mathcal{J}_n(\Gamma), \mathbf{s}^{(R)})$ expresses R , *or* there exists an n -ary polymorphism of Γ which does not preserve R at $\mathbf{s}^{(R)}$. Thus we get the following theoretical upper bounds to the size of a witness to the expressibility or inexpressibility of R from Γ .

Corollary 1 ([9,11,15]). *Let $\Gamma \cup \{R\}$ be a set of relations on D , and let $n = |R|$.*

1. *If R is expressible from Γ , then R can be expressed by a CSP instance (or a primitive positive formula) with variable set of size $\leq |D|^n$.*
2. *R is not expressible from Γ if and only if there exists a polymorphism of Γ of arity $\leq n$ which does not preserve R .*

Example 3. Consider again the example in Section 1. The relation $V = \{3, 4, 5\}$ on the 6-element domain $\{0, 1, 2, 3, 4, 5\}$ is expressible from $\Gamma = \{\rightarrow, U\}$, so Corollary 1 promises a CSP witness having $\leq 6^3 = 216$ variables. Conversely, the complement \nrightarrow of \rightarrow turns out to be not expressible from Γ . Since \nrightarrow has 26 rows, Corollary 1 promises a witnessing polymorphism of arity ≤ 26 .

Note the ridiculousness of the bounds in Example 3. Corollary 1 guarantees a CSP instance having ≤ 216 variables to express V , when in fact we have an instance using just 4 variables. Even worse is the promise of a 26-ary polymorphism witnessing the inexpressibility of \neg ; just storing the values of a random 26-ary function on $\{0, 1, 2, 3, 4, 5\}$ would require over 5×10^8 terabytes. Yet the 1-ary polymorphism of Example 2 fails to preserve \neg (e.g., at $(2, 2)$) and so already witnesses its inexpressibility.

Example 3 illustrates the fact that the upper bounds to the sizes of witnesses guaranteed by Corollary 1 are exponential in the size of the test relation. It is natural to ask if these upper bounds can be improved. For example, Cohen and Jeavons [5, p. 313] pose as an open research question the identification of circumstances under which sub-exponential sized CSP instances can be found witnessing expressible relations. Our first theorem says “not always”:

Theorem 1. *For infinitely many n there exist a constraint language Γ_n and a relation R_n , both on a 22-element domain, such that $|R_n| = n$, R_n is expressible from Γ_n , but every CSP(Γ_n) instance expressing R_n has at least $2^{n/3}$ variables.*

Dually, our next theorem shows that in general we cannot hope to detect inexpressibility with sub-exponential sized polymorphisms.

Theorem 2. *For infinitely many n there exist a constraint language Γ'_n and a relation R'_n , both on a 22-element domain, such that $|R'_n| = n$, R'_n is not expressible from Γ'_n , but every witnessing polymorphism has arity at least $n/3$.*

We formally define **EXPR** to be the combinatorial decision problem which takes as input a triple (D, Γ, R) (where D is a finite domain, Γ is a finite constraint language on D , and R is another relation on D), and asks whether R is expressible from Γ . **EXPR** has also been called \exists -**INVSAT** (the *existential inverse satisfiability problem*) [76] and the *pp-definability problem* [4].

Corollary 1 and the discussion preceding it give a general algorithm for testing \neg -**EXPR**: among all functions $h : D^n \rightarrow D$ where $n = |R|$, search for one which (i) is a polymorphism of Γ , and (ii) does not preserve R at $\mathbf{s}^{(R)}$. This naive algorithm puts **EXPR** in **co-NEXPTIME**. Dalmau [7, p. 163] speculated that perhaps there exists a better, more sophisticated algorithm which would place **EXPR** in a lower complexity class. Suggestively, Creignou *et al* [6] have proved that **EXPR** restricted to the boolean domain is in **P**.

At a workshop at AIM in 2008, a working group led by M. Vardi contrarily conjectured that there is essentially no algorithm better than the naive one, in the sense that **EXPR** restricted to 3-element domains is **co-NEXPTIME**-complete [4]. In our last theorem we very nearly confirm this conjecture:

Theorem 3. *There exists $d > 1$ such that **EXPR** restricted to d -element domains is **co-NEXPTIME**-complete.*

The remainder of this paper is devoted to proving Theorems 1–3 via an interpretation of certain tiling problems defined by domino systems.

3 Domino Systems and Tiling Problems

A *tiling problem* is a particular kind of constraint satisfaction problem whose constraints are organized “horizontally and vertically.” More precisely:

Definition 3 ([10,2]). A domino system is a triple $\mathcal{D} = (\Delta, H, V)$ where Δ is a finite nonempty set (of “tile types”) and H, V are binary relations on Δ (called the horizontal and vertical adjacency constraint relations).

Notation 4. For $N > 1$ we will use $[N \times N]$ to denote the set

$$[N \times N] = \{(i, j) : i, j \in \mathbb{Z}, 0 \leq i, j < N\}.$$

We informally identify the element $(i, j) \in [N \times N]$ with the unit square in the x - y plane whose lower-left corner has coordinates (i, j) . The k th row of $[N \times N]$ is the subset $\text{Row}_k = \{(i, k) : 0 \leq i < N\}$, while the k th column is the subset $\text{Col}_k = \{(k, j) : 0 \leq j < N\}$. Figure 2 illustrates the board $[4 \times 4]$.

Definition 5. Suppose $\mathcal{D} = (\Delta, H, V)$ is a domino system and $N > 1$. A tiling of $[N \times N]$ by \mathcal{D} is a mapping $\tau : [N \times N] \rightarrow \Delta$ assigning to each square $(i, j) \in [N \times N]$ a tile type $\tau[i, j] \in \Delta$, subject to the following constraints:

- For each pair $(i, j), (i+1, j)$ of horizontally adjacent squares in $[N \times N]$, the corresponding pair $(\tau[i, j], \tau[i+1, j])$ of tile types satisfies H .
- For each pair $(i, j), (i, j+1)$ of vertically adjacent squares in $[N \times N]$, the corresponding pair $(\tau[i, j], \tau[i, j+1])$ of tile types satisfies V .

Example 4. Define a domino system $\mathcal{D}_1 = (\Delta, H, V)$ where

$$\begin{aligned} \Delta &= \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}\} \\ H &= \{(\mathbf{a}, \mathbf{b}), (\mathbf{b}, \mathbf{a}), (\mathbf{b}, \mathbf{d}), (\mathbf{c}, \mathbf{b}), (\mathbf{d}, \mathbf{c}), (\mathbf{d}, \mathbf{f}), (\mathbf{e}, \mathbf{b})\} \\ V &= \{(\mathbf{a}, \mathbf{b}), (\mathbf{a}, \mathbf{e}), (\mathbf{b}, \mathbf{b}), (\mathbf{b}, \mathbf{c}), (\mathbf{c}, \mathbf{d}), (\mathbf{d}, \mathbf{d}), (\mathbf{e}, \mathbf{e}), (\mathbf{f}, \mathbf{f})\}. \end{aligned}$$

The map $\tau : [4 \times 4] \rightarrow \Delta$ pictured in Figure 2 is a tiling of $[4 \times 4]$ by \mathcal{D}_1 .

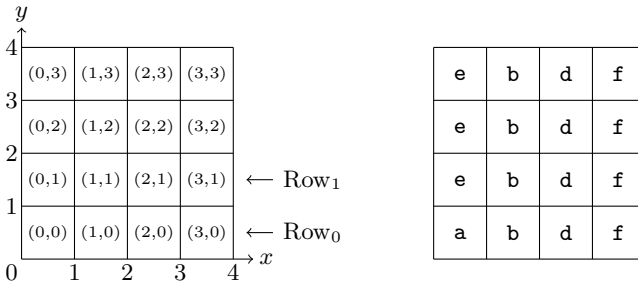


Fig. 2. The board $[4 \times 4]$ and one tiling of it by \mathcal{D}_1

We need to be able to discuss partial tilings and tilings with initial conditions.

Definition 6. Suppose $\mathcal{D} = (\Delta, H, V)$ is a domino system and $N > 1$.

1. Let $\mathbf{w} = (w_0, \dots, w_{m-1}) \in \Delta^m$ with $0 < m \leq N$, and let $j < N$. A tiling τ of $[N \times N]$ by \mathcal{D} satisfies the initial condition \mathbf{w} if $\tau[i, 0] = w_i$ for all $i < m$.
2. If $U \subseteq [N \times N]$ then we may speak of tilings of U by \mathcal{D} satisfying \mathbf{w} ; these are mappings from U to Δ which satisfy those horizontal, vertical and initial condition constraints that mention squares in U only.
3. Given a tiling τ of $[N \times N]$ by \mathcal{D} , we say that τ has a repeated row if there exists $\mathbf{z} \in \Delta^N$ and distinct $j < k < N$ such that τ makes the same assignment to Row_j and to Row_k ; that is, $\tau[i, j] = \tau[i, k]$ for all $0 \leq i < N$.

Example 3 (continued). The tiling of $[4 \times 4]$ pictured in Figure 2 satisfies the initial condition (a, b). However, \mathcal{D}_1 cannot tile $[4 \times 4]$ with initial condition (b, a).

In this paper we will be particularly interested in the following “exponential tiling problem,” which we define in both local and uniform versions.

Definition 7. 1. Given a domino system $\mathcal{D} = (\Delta, H, V)$, $\text{EXPTILE}(\mathcal{D})$ denotes the combinatorial decision problem whose input is a triple $(\mathcal{D}, m, \mathbf{w})$ where $m \geq 1$ and $\mathbf{w} \in \Delta^m$, and which asks whether \mathcal{D} tiles $[2^m \times 2^m]$ with initial condition \mathbf{w} .

2. $\text{EXPTILE} = \bigcup_{\mathcal{D}} \text{EXPTILE}(\mathcal{D})$.

3.1 A Domino System That Exponentially Counts

Our proofs of Theorems 1 and 2 will exploit the following fact.

Proposition 2. *There exists a domino system $\mathcal{D}_e = (\Delta_e, H_e, V_e)$ with the following property: for all $m > 2$ there exist m -tuples $\mathbf{w}_m, \mathbf{w}'_m \in (\Delta_e)^m$ such that*

1. \mathcal{D}_e does not tile $[2^m \times 2^m]$ with initial condition \mathbf{w}_m , but \mathcal{D}_e does tile U with initial condition \mathbf{w}_m for every $U \subseteq [2^m \times 2^m]$ satisfying $|U| < 2^m$.
2. \mathcal{D}_e tiles $[2^m \times 2^m]$ with initial condition \mathbf{w}'_m , and moreover every tiling of $[2^m \times 2^m]$ by \mathcal{D}_e with initial condition \mathbf{w}'_m has no repeated row.

We describe one way to construct such a domino system \mathcal{D}_e . Our strategy is to design \mathcal{D}_e so that its tilings of subsets of $[2^m \times 2^m]$ force consecutive rows to encode consecutive integers between 0 and $2^m - 1$.

If $m > 0$ and $x \in \{0, 1, 2, 3, \dots, 2^m - 1\}$, let $\text{Bin}_m(x)$ denote the reverse m -bit binary representation of x (least significant bit at the left).

Example 5. $\text{Bin}_5(6) = (0, 1, 1, 0, 0)$.

We define some sets of new symbols; they will be the tile types for \mathcal{D}_e :

$$\begin{aligned} \Delta_0 &= \{0_L^-, 0_M^-, 0_M^+, 0_R^-, 0_R^+\} & \Delta_1 &= \{1_L^\diamond, 1_M^\diamond, 1_M^+, 1_R^\diamond, 1_R^+\} \\ \Delta_{01} &= \Delta_0 \cup \Delta_1 & \Delta_e &= \Delta_{01} \cup \{\triangleleft\}. \end{aligned}$$

Definition 8. Suppose $m > 2$ and $x \in \{0, 1, 2, 3, \dots, 2^m - 1\}$, with $\text{Bin}_m(x) = (b_0, b_1, \dots, b_{m-1})$. The annotated m -bit binary representation of x is the m -tuple $\text{AnnBin}_m(x) = (\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_{m-1}) \in (\Delta_{01})^m$ given as follows: $\mathbf{a}_i = (b_i)_X^s$ where

- X is L if $i = 0$, R if $i = m - 1$, and M otherwise.
- If there exists $j < i$ such that $b_j = 1$, then s is $+$. Otherwise, s is $-$ if $b_i = 0$ while s is \diamond if $b_i = 1$.

Example 6. $\text{AnnBin}_5(6) = (0_L^-, 1_M^\diamond, 1_M^+, 0_M^+, 0_R^+)$.

Note that the “bases” of the entries of $\text{AnnBin}_m(x)$ give the reverse m -bit binary representation of x ; the subscripts are exactly (L, M, \dots, M, R) ; and the superscripts are one of the following patterns: $(\diamond, +, \dots, +)$, $(-, \dots, -, \diamond, +, \dots, +)$, $(-, \dots, -, \diamond)$, or $(-, -, \dots, -)$, where \diamond occurs at the first bit of x equalling 1.

Fix $m > 2$ and define τ_m to be the mapping $[2^m \times 2^m] \rightarrow \Delta_e$ which for each $0 \leq j < 2^m$ assigns $\text{AnnBin}_m(j)$ to the first m entries in Row_j , and assigns \triangleleft to all remaining squares (see Figure 3).

Row ₁₅	1_L^\diamond	1_M^+	1_M^+	1_R^+	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Row ₅	1_L^\diamond	0_M^+	1_M^+	0_R^+	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
Row ₄	0_L^-	0_M^-	1_M^\diamond	0_R^+	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
Row ₃	1_L^\diamond	1_M^+	0_M^+	0_R^+	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
Row ₂	0_L^-	1_M^\diamond	0_M^+	0_R^+	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
Row ₁	1_L^\diamond	0_M^+	0_M^+	0_R^+	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft
Row ₀	0_L^-	0_M^-	0_M^-	0_R^-	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft	\triangleleft

Fig. 3. τ_4 defined on $[16 \times 16]$

Now let $\mathcal{D}_e = (\Delta_e, H_e, V_e)$ be the smallest domino system with respect to which τ_4 is a tiling of $[16 \times 16]$. That is, define

$$\begin{aligned}
 H_e &= \{0_L^-\} \times \{0_M^-, 1_M^\diamond\} \cup \{1_L^\diamond\} \times \{0_M^+, 1_M^+\} \cup \{0_M^-\} \times \{0_M^-, 1_M^\diamond, 0_R^-, 1_R^\diamond\} \\
 &\quad \cup \{0_M^+, 1_M^+, 1_M^\diamond\} \times \{0_M^+, 1_M^+, 0_R^+, 1_R^+\} \cup \{0_R^-, 0_R^+, 1_R^\diamond, 1_R^+\} \times \{\triangleleft\} \\
 V_e &= \{(0_L^-, 1_L^\diamond), (1_L^\diamond, 0_L^-), (0_M^-, 0_M^+), (0_M^+, 0_M^+), (0_M^+, 1_M^\diamond), (1_M^\diamond, 1_M^+), (1_M^+, 1_M^+), \\
 &\quad (1_M^+, 0_M^-), (0_R^-, 0_R^+), (0_R^+, 0_R^+), (0_R^+, 1_R^\diamond), (1_R^\diamond, 1_R^+), (1_R^+, 1_R^+), (\triangleleft, \triangleleft)\}.
 \end{aligned}$$

The reader can check that \mathcal{D}_e , thus defined, satisfies Proposition 2 with $\mathbf{w}_m = \text{AnnBin}_m(1)$ and $\mathbf{w}'_m = \text{AnnBin}_m(0)$. Indeed, τ_m is the unique tiling by \mathcal{D}_e of $[2^m \times 2^m]$ with initial condition \mathbf{w}'_m , and clearly τ_m has no repeated rows. On the other hand, \mathcal{D}_e cannot tile $[2^m \times 2^m]$ with initial condition \mathbf{w}_m (as it cannot

count past $2^m - 1$), but if $U \subseteq [2^m \times 2^m]$ with $|U| < 2^m$, then there must exist $k < 2^m$ such that U is disjoint from Row_k . In this case \mathcal{D}_e can easily tile U with initial condition \mathbf{w}_m , simply by assigning $\text{AnnBin}_m(j+1)$ to the first m entries of Row_j for each $j < k$, assigning $\text{AnnBin}_m(j)$ to the first m entries of Row_j for all $k < j < 2^m$, and \triangleleft to all remaining entries.

4 Interpreting Exponential Tiling into Expressibility

In this section we will describe the main (and most difficult) construction of this paper. It takes as input an instance $(\mathcal{D}, m, \mathbf{w})$ of EXPTILE where $m > 2$ and m is a power of 2, and produces as output an instance (D, Γ, R) of EXPR , so that

R is expressible from $\Gamma \Leftrightarrow \mathcal{D}$ cannot tile $[2^m \times 2^m]$ with initial condition \mathbf{w} .

Furthermore, the existence of “small” witnesses to the expressibility or inexpressibility of R will be connected to the existence of “small” witnesses to untilability or tilability (small subsets of $[2^m \times 2^m]$ that cannot be tiled, or tilings of $[2^m \times 2^m]$ with repeated rows). Thus Proposition 2 will give us Theorems 1 and 2. Because we also wish the construction $(\mathcal{D}, m, \mathbf{w}) \mapsto (D, \Gamma, R)$ to give a logspace reduction of this fragment of EXPTILE into $\neg\text{EXPR}$, the sizes of D , Γ , and the relations in $\Gamma \cup \{R\}$ must be bounded by a polynomial in $|\Delta| + m$, and the construction itself must be executable in logspace in $|\Delta| + m$.

4.1 Defining the Domain D and Encoding $[2^m \times 2^m]$ in D^m

For the remainder of Section 4 we fix a domino system $\mathcal{D} = (\Delta, H, V)$, an integer $m = 2^t$ ($t > 1$), and an m -tuple $\mathbf{w} = (w_0, w_1, \dots, w_{m-1}) \in \Delta^m$.

Definition 9. *The domain D for our constraint language is the disjoint union of the sets Δ , $P := \{\mathbf{p}_{00}, \mathbf{p}_{01}, \mathbf{p}_{10}, \mathbf{p}_{11}\}$, $\{0, 1\}$, $\{\mathbf{a}, \mathbf{b}\}$, $\{\top, \perp\}$, and $\{\infty\}$.*

We next explain how we will interpret $[2^m \times 2^m]$ in D^m . For $(x, y) \in [2^m \times 2^m]$, write $\text{Bin}_m(x) = (x_0, x_1, \dots, x_{m-1})$ and $\text{Bin}_m(y) = (y_0, y_1, \dots, y_{m-1})$, the reverse m -bit binary representations of x and y respectively, and let $\mathbf{p}(x, y) \in D^m$ be given by $\mathbf{p}(x, y)[i] = \mathbf{p}_{x_i y_i}$ for $0 \leq i < m$. In this way the elements of $[2^m \times 2^m]$ are put in one-to-one correspondence with the elements of P^m .

Example 7. If $m = 8$, then $\mathbf{p}(53, 188) = (\mathbf{p}_{10}, \mathbf{p}_{00}, \mathbf{p}_{11}, \mathbf{p}_{01}, \mathbf{p}_{11}, \mathbf{p}_{11}, \mathbf{p}_{00}, \mathbf{p}_{01})$.

Next we define $t + 1$ auxiliary elements $\beta_0, \beta_1, \dots, \beta_{t-1}, \gamma$ in D^m (recall that $t = \log_2 m$), first by example. If $m = 8$ (so $t = 3$), then

$$\begin{aligned}\beta_0 &= (0, 1, 0, 1, 0, 1, 0, 1) \\ \beta_1 &= (0, 0, 1, 1, 0, 0, 1, 1) \\ \beta_2 &= (0, 0, 0, 0, 1, 1, 1, 1) \\ \gamma &= (\mathbf{b}, \mathbf{b}, \mathbf{a}, \mathbf{b}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{b}).\end{aligned}$$

Note that the columns on the right-hand side of the above equations, restricted to the β_i 's, are $\text{Bin}_3(0), \text{Bin}_3(1), \text{Bin}_3(2), \dots, \text{Bin}_3(7)$ respectively. In general,

Definition 10

1. $\beta_0, \dots, \beta_{t-1} \in \{0, 1\}^m$ are defined so that $(\beta_0[i], \beta_1[i], \dots, \beta_{t-1}[i]) = \text{Bin}_t(i)$ for all $0 \leq i < m$.
2. The element $\gamma \in \{\mathbf{a}, \mathbf{b}\}^m$ is defined by $\gamma[i] = \mathbf{b}$ if $i = 2^k - 1$ for some $k \leq t$, and $\gamma[i] = \mathbf{a}$ otherwise.
3. $\mathbf{s} = (\beta_0, \beta_1, \dots, \beta_{t-1}, \gamma) \in (D^m)^{t+1}$.
4. $R_0 = \text{proj}(\mathbf{s}) = \{(\text{Bin}_t(i), \gamma[i]) : 0 \leq i < m\}$.

Example 8. If $m = 8$, then $R_0 = \{(0, 0, 0, \mathbf{b}), (1, 0, 0, \mathbf{b}), (0, 1, 0, \mathbf{a}), (1, 1, 0, \mathbf{b}), (0, 0, 1, \mathbf{a}), (1, 0, 1, \mathbf{a}), (0, 1, 1, \mathbf{a}), (1, 1, 1, \mathbf{b})\}$.

The elements $\beta_0, \dots, \beta_{t-1}, \gamma \in D^m$ and the relation R_0 will help us coordinatize P^m . The element γ helps to enforce some “rigidity” as explained in the next lemma.

Lemma 1. *Suppose σ is a self-map from $\{0, 1, \dots, t-1\}$ to itself, and $\mathbf{d} = (\beta_{\sigma(0)}, \beta_{\sigma(1)}, \dots, \beta_{\sigma(t-1)}, \gamma)$. If $\text{proj}(\mathbf{d}) \subseteq R_0$, then $\sigma(i) = i$ for all $i < t$.*

Once the constraint language Γ has been constructed, we will be intensely interested in the $(t+1)$ -ary relation S expressed by $(\mathcal{J}_m(\Gamma), \mathbf{s})$. This relation is equivalently defined as the set of images of $(\beta_0, \dots, \beta_{t-1}, \gamma)$ under the m -ary polymorphisms of Γ . We will be particularly interested in learning whether the $(t+1)$ -tuple $(\top, \top, \dots, \top)$ belongs to S . Call a map $f : D^m \rightarrow D$ *special* if it satisfies $f(\beta_0) = f(\beta_1) = \dots = f(\beta_{t-1}) = f(\gamma) = \top$. The intermediate aim of the construction of Γ is to achieve the following two competing goals:

1. If $h : D^m \rightarrow D$ is any special m -ary polymorphism of Γ , then h should map P^m to Δ ; moreover, the restriction of h to P^m should encode a tiling of $[2^m \times 2^m]$ by \mathcal{D} with initial condition \mathbf{w} .
2. Conversely, if τ is any tiling by \mathcal{D} of $[2^m \times 2^m]$ with initial condition \mathbf{w} , then there should exist a special m -ary polymorphism h of Γ whose restriction to P^m encodes τ .

An immediate consequence of these goals, when achieved, is that the expressible relation S will contain the constant tuple $(\top, \top, \dots, \top)$ if and only if \mathcal{D} tiles $[2^m \times 2^m]$ with initial condition \mathbf{w} . This will somehow help us in achieving the goals described at the beginning of Section 4.

4.2 Defining the Constraint Language Γ and the Test Relation R

Each relation in Γ will be constructed using the following recipe. Fix $k = 1$ or 2. Choose a k -ary relation \mathcal{H} on P^m and a k -ary relation C on Δ , subject to the requirement that \mathcal{H} factors as an m -fold product relation $\mathcal{H} = H_0 \times H_1 \times \dots \times H_{m-1}$ for some k -ary relations H_0, H_1, \dots, H_{m-1} on P . Then define the $(k+t+1)$ -ary relation $\mathcal{R}_{\mathcal{H} \Rightarrow C}$ on D as follows:

$$\begin{aligned} \mathcal{R}_{\mathcal{H} \Rightarrow C} = & \bigcup_{i=0}^{m-1} \{(\mathbf{x}, \mathbf{y}) \in P^k \times (\{0, 1\}^t \times \{\mathbf{a}, \mathbf{b}\}) : \mathbf{x} \in H_i, \mathbf{y} = (\text{Bin}_t(i), \gamma[i])\} \\ & \cup \{(\mathbf{x}, \mathbf{y}) \in \Delta^k \times \{\top, \perp\}^{t+1} : \perp \in \{\mathbf{y}[0], \dots, \mathbf{y}[t]\} \text{ or } \mathbf{x} \in C\} \\ & \cup \{(\infty, \infty, \dots, \infty)\}. \end{aligned}$$

Lemma 2. *For any relation $\mathcal{R}_{\mathcal{H} \Rightarrow C}$ constructed according to the recipe above:*

1. $\mathcal{R}_{\mathcal{H} \Rightarrow C} \subseteq (P^k \times \{0, 1\}^t \times \{\mathbf{a}, \mathbf{b}\}) \cup (\Delta^k \times \{\top, \perp\}^{t+1}) \cup \{\infty\}^{k+t+1}$.
2. For any $\mathbf{c} \in (D^m)^k$, $\text{proj}(\mathbf{c}, \beta_0, \beta_1, \dots, \beta_{t-1}, \gamma) \subseteq \mathcal{R}_{\mathcal{H} \Rightarrow C}$ if and only if $\mathbf{c} \in \mathcal{H}$.
3. For any $\mathbf{c} \in D^k$, $(\mathbf{c}, \top, \top, \dots, \top) \in \mathcal{R}_{\mathcal{H} \Rightarrow C}$ if and only if $\mathbf{c} \in C$.

Our first family of relations will encode the adjacency constraints of \mathcal{D} .

Definition 11. 1. For an integer $0 < x < 2^m$ define $\text{lg}(x)$ to be the largest integer $0 \leq k < m$ such that 2^k divides x .

2. For $0 \leq k < m$ let $\mathcal{H}\mathcal{A}^{(k)}, \mathcal{V}\mathcal{A}^{(k)}$ be the following binary relations on P^m :

$$\begin{aligned} \mathcal{H}\mathcal{A}^{(k)} &= \{(\mathbf{p}(x, y), \mathbf{p}(x+1, y)) : 0 \leq x, y < 2^m, x \neq 2^m-1, \text{lg}(x+1) = k\} \\ \mathcal{V}\mathcal{A}^{(k)} &= \{(\mathbf{p}(x, y), \mathbf{p}(x, y+1)) : 0 \leq x, y < 2^m, y \neq 2^m-1, \text{lg}(y+1) = k\}. \end{aligned}$$

I.e., $\mathcal{H}\mathcal{A}^{(k)}$ is the binary relation on P^m encoding those pairs $((x, y), (x+1, y))$ of horizontally adjacent elements of $[2^m \times 2^m]$ for which the reverse binary representation of x begins with k 1's followed by 0. The reader should verify that each of the relations $\mathcal{H}\mathcal{A}^{(k)}, \mathcal{V}\mathcal{A}^{(k)}$ factors as an m -fold product relation.

Example 9. If $m = 8$ and $k = 3$, then

$$\begin{aligned} \mathcal{H}\mathcal{A}^{(3)} &= \{(\mathbf{p}_{10}, \mathbf{p}_{00}), (\mathbf{p}_{11}, \mathbf{p}_{01})\}^3 \times \{(\mathbf{p}_{00}, \mathbf{p}_{10}), (\mathbf{p}_{01}, \mathbf{p}_{11})\} \\ &\quad \times \{(\mathbf{p}_{00}, \mathbf{p}_{00}), (\mathbf{p}_{01}, \mathbf{p}_{01}), (\mathbf{p}_{10}, \mathbf{p}_{10}), (\mathbf{p}_{11}, \mathbf{p}_{11})\}^4. \end{aligned}$$

Definition 12. Recall that $\mathcal{D} = (\Delta, H, V)$. The set of adjacency relations is

$$\mathcal{A} = \{\mathcal{R}_{\mathcal{H}\mathcal{A}^{(k)} \Rightarrow H} : 0 \leq k < m\} \cup \{\mathcal{R}_{\mathcal{V}\mathcal{A}^{(k)} \Rightarrow V} : 0 \leq k < m\}.$$

For each $(x, y) \in [2^m \times 2^m]$, the singleton unary relation $\{\mathbf{p}(x, y)\}$ on P^m clearly factors as an m -fold product relation.

Definition 13. Recall that $\mathbf{w} = (w_0, \dots, w_{m-1})$. The set of initial relations is

$$\mathcal{I} = \{\mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}} : 0 \leq k < m\}.$$

Definition 14. Our constraint language is $\Gamma = \mathcal{A} \cup \mathcal{I} \cup \{\mathcal{R}_{P^n \Rightarrow \Delta}\}$.

Finally, we define two further $(t+1)$ -ary relations on D . The first relation, R , is an easily constructed relation whose expressibility from Γ will be our chief interest; it may be informally defined as $\mathcal{R}_{\top \Rightarrow \perp}$ where \top and \perp are here being used to denote the 0-ary “true” and “false” relations on P^m and Δ respectively. The second relation, S , is easily defined but not easily constructed and is not claimed to be part of the output of our logspace construction.

Definition 15. Recall that $R_0 = \text{proj}(\mathbf{s})$ where $\mathbf{s} = (\beta_0, \beta_1, \dots, \beta_{t-1}, \gamma)$.

$$R = R_0 \cup (\{\top, \perp\}^{t+1} \setminus \{(\top, \top, \dots, \top)\}) \cup \{(\infty, \infty, \dots, \infty)\}$$

$$S = \{(h(\beta_0), h(\beta_1), \dots, h(\beta_{t-1}), h(\gamma)) : h \text{ is an } m\text{-ary polymorphism of } \Gamma\}.$$

4.3 Connecting Polymorphisms, Tilings, and Expressibility

For convenience, define the notation $\widehat{\top} = (\top, \top, \dots, \top)$ and $\widehat{\infty} = (\infty, \infty, \dots, \infty)$.

Lemma 3. 1. S is the smallest $(t + 1)$ -ary relation expressible from Γ and containing R_0 .

2. $R \subseteq S \subseteq R \cup \{\widehat{\top}\}$.

3. R is expressible from Γ if and only if $\widehat{\top} \notin S$.

Proof. $S = \pi_{\mathbf{s}}(\text{Sol}(\mathcal{J}_m(\Gamma)))$, i.e., S is the relation expressed by $(\mathcal{J}_m(\Gamma), \mathbf{s})$ where $\mathbf{s} = (\beta_0, \dots, \beta_{t-1}, \gamma)$. (1) follows from this observation, the definition of R_0 , and Proposition [1](#). To prove $S \subseteq R \cup \{\widehat{\top}\}$, it thus suffices to show that $R \cup \{\widehat{\top}\}$ is expressible from Γ (as it clearly contains R_0). This is easy, since the primitive positive formula $\exists z \mathcal{R}_{P \Rightarrow \Delta}(z, x_0, x_1, \dots, x_t)$ defines $R \cup \{\widehat{\top}\}$. As (3) follows from (1) and (2), it remains only to prove $R \subseteq S$.

Clearly $R_0 \subseteq S$ by (1), and $\widehat{\infty} \in S$ since the constant function $D^m \rightarrow \{\infty\}$ is a polymorphism of Γ . Suppose now that $\mathbf{f} = (f_0, \dots, f_t) \in \{\top, \perp\}^{t+1} \setminus \{\widehat{\top}\}$. Pick any $d_0 \in \Delta$ and define $h_{\mathbf{f}} : D^m \rightarrow D$ by

$$h_{\mathbf{f}}(\mathbf{x}) = \begin{cases} d_0 & \text{if } \mathbf{x} \in P^m \\ f_i & \text{if } \mathbf{x} = \beta_i \text{ for some } i < t \\ f_t & \text{if } \mathbf{x} = \gamma \\ \perp & \text{if } \mathbf{x} \in \{0, 1\}^m \cup \{\mathbf{a}, \mathbf{b}\}^m \setminus \{\beta_0, \dots, \beta_{t-1}, \gamma\} \\ \infty & \text{otherwise.} \end{cases}$$

To prove $\mathbf{f} \in S$, it suffices to show that $h_{\mathbf{f}}$ is a polymorphism of Γ . We will show simply that $h_{\mathbf{f}}$ preserves each initial relation $\mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}}$ at all $(t + 2)$ -tuples in D^m , the proofs for the other relations being similar. Indeed, if this were false, then there would exist $\mathbf{c} = (\mathbf{x}, \mathbf{z}_0, \dots, \mathbf{z}_t) \in (D^m)^{t+2}$ with

- (a) $\text{proj}(\mathbf{c}) \subseteq \mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}}$, but
- (b) $(h_{\mathbf{f}}(\mathbf{x}), h_{\mathbf{f}}(\mathbf{z}_0), \dots, h_{\mathbf{f}}(\mathbf{z}_t)) \notin \mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}}$.

At least one of $h_{\mathbf{f}}(\mathbf{x}), h_{\mathbf{f}}(\mathbf{z}_0), \dots, h_{\mathbf{f}}(\mathbf{z}_t)$ must be different from ∞ . Hence by definition of $h_{\mathbf{f}}$, $\{\mathbf{x}, \mathbf{z}_0, \dots, \mathbf{z}_t\}$ is not disjoint from $P^m \cup \{0, 1\}^m \cup \{\mathbf{a}, \mathbf{b}\}^m$. This last fact, Lemma [2](#)(1), and item (a) above then yield $\mathbf{x} \in P^m$, $\mathbf{z}_0, \dots, \mathbf{z}_{t-1} \in \{0, 1\}^m$, and $\mathbf{z}_t \in \{\mathbf{a}, \mathbf{b}\}^m$. Hence $(h_{\mathbf{f}}(\mathbf{x}), h_{\mathbf{f}}(\mathbf{z}_0), \dots, h_{\mathbf{f}}(\mathbf{z}_t)) = (d_0, f'_0, \dots, f'_t)$ for some $f'_0, \dots, f'_t \in \{\top, \perp\}$ (by the definition of $h_{\mathbf{f}}$). If $d_0 = w_k$ or at least one f'_i is \perp , then clearly $(d_0, f'_0, \dots, f'_t) \in \mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}}$; hence $d_0 \neq w_k$ and all f'_i are \top . The definition of $h_{\mathbf{f}}$ then implies that $\mathbf{z}_t = \gamma$ and there exists a selfmap σ on $\{0, 1, \dots, t-1\}$ such that $\mathbf{z}_i = \beta_{\sigma(i)}$ for $i < t$. Lemma [1](#) then implies that $\sigma(i) = i$ for all $i < t$, so $\mathbf{c} = (\mathbf{x}, \beta_0, \dots, \beta_{t-1}, \gamma)$ with $\mathbf{x} \in P^m$. The definition of $h_{\mathbf{f}}$ then gives $(d_0, \top, \dots, \top) = (d_0, f_0, \dots, f_t)$, contradicting the assumption that $\mathbf{f} \neq \widehat{\top}$. \square

We can now prove the desired connection between tilings and expressibility.

Proposition 3. *The following are equivalent:*

1. R is not expressible from Γ .
2. $\widehat{\top} \in S$.
3. \mathcal{D} tiles $[2^m \times 2^m]$ with initial condition \mathbf{w} .

Proof. (1) \Leftrightarrow (2) follows from Lemma 3.

(2) \Rightarrow (3). Assume $\widehat{\top} \in S$; choose an m -ary polymorphism h of Γ satisfying $h(\beta_0) = \dots = h(\beta_{t-1}) = h(\gamma) = \top$. We first show that h maps P^m into Δ . Indeed, let $\mathbf{x} \in P^m$; then $\text{proj}((\mathbf{x}, \beta_0, \dots, \beta_{t-1}, \gamma)) \subseteq \mathcal{R}_{P^m \Rightarrow \Delta}$ by Lemma 2(2). As h is a polymorphism of Γ , it preserves $\mathcal{R}_{P^m \Rightarrow \Delta}$ at $(\mathbf{x}, \beta_0, \dots, \beta_{t-1}, \gamma)$; hence we get $(h(\mathbf{x}), h(\beta_0), \dots, h(\beta_{t-1}), h(\gamma)) \in \mathcal{R}_{P^m \Rightarrow \Delta}$, i.e., $(h(\mathbf{x}), \top, \dots, \top) \in \mathcal{R}_{P^m \Rightarrow \Delta}$. This with Lemma 2(3) implies $h(\mathbf{x}) \in \Delta$, as claimed.

Thus we may define a map $\tau_h : [2^m \times 2^m] \rightarrow \Delta$ by $\tau_h[i, j] = h(\mathbf{p}(i, j))$. Using the fact that h preserves the adjacency and initial relations at all tuples of the form $(\mathbf{x}, \mathbf{x}', \beta_0, \dots, \beta_{t-1}, \gamma)$ or $(\mathbf{x}, \beta_0, \dots, \beta_{t-1}, \gamma)$ respectively (\mathbf{x}, \mathbf{x}' varying over P^m), and using Lemma 2(2,3), one can show that τ_h is a tiling of $[2^m \times 2^m]$ with initial condition \mathbf{w} .

(3) \Rightarrow (2). Assume that τ is a tiling of $[2^m \times 2^m]$ by \mathcal{D} with initial condition \mathbf{w} . Define $h_\tau : D^m \rightarrow D$ by

$$h_\tau(\mathbf{x}) = \begin{cases} \tau[i, j] & \text{if } \mathbf{x} = \mathbf{p}(i, j) \text{ where } (i, j) \in [2^m \times 2^m] \\ \top & \text{if } \mathbf{x} \in \{\beta_0, \dots, \beta_{t-1}, \gamma\} \\ \perp & \text{if } \mathbf{x} \in \{0, 1\}^m \cup \{\mathbf{a}, \mathbf{b}\}^m \setminus \{\beta_0, \dots, \beta_{t-1}, \gamma\} \\ \infty & \text{otherwise.} \end{cases}$$

It suffices to prove that h_τ is a polymorphism of Γ . We repeat the proof that h_f preserves $\mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}}$ in the proof of Lemma 3, replacing h_f with h_τ . Again, we suppose for the sake of contradiction that we have $\mathbf{c} = (\mathbf{x}, \mathbf{z}_0, \dots, \mathbf{z}_t) \in (D^m)^{t+2}$ with

- (a) $\text{proj}(\mathbf{c}) \subseteq \mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}}$, but
- (c) $(h_\tau(\mathbf{x}), h_\tau(\mathbf{z}_0), \dots, h_\tau(\mathbf{z}_t)) \notin \mathcal{R}_{\{\mathbf{p}(k,0)\} \Rightarrow \{w_k\}}$.

Arguing as before, we get

- (d) $\mathbf{c} = (\mathbf{x}, \beta_0, \dots, \beta_{t-1}, \gamma)$, and
- (e) $\mathbf{x} \in P^m$ and $h_\tau(\mathbf{x}) \neq w_k$.

Items (a) and (d), with Lemma 2, imply $\mathbf{x} = \mathbf{p}(k, 0)$. Hence $h_\tau(\mathbf{x}) = \tau[k, 0]$, which with item (e) contradicts the fact that τ satisfies \mathbf{w} at Row_0 . \square

As $|R| = 3m$, Corollary 1 implies that if R is expressible from Γ then R can be expressed by a $\text{CSP}(\Gamma)$ instance having $|D|^{3m}$ variables, while if R is not expressible from Γ then this is witnessed by a polymorphism of Γ of arity $3m$. We can slightly improve this. On the one hand, Lemma 3 clearly implies:

Corollary 2. *If R is not expressible from Γ , then this is witnessed by an m -ary polymorphism.*

Conversely, a careful examination of the proof of Proposition 3(2) \Rightarrow (3) shows that the only constraints on h needed to complete the proof are ones involving the values of h at elements of $P^m \cup \{\beta_0, \dots, \beta_{t-1}, \gamma\}$. Hence:

Corollary 3. *If R is expressible from Γ , then it can be expressed by an instance of $\text{CSP}(\Gamma)$ (or a primitive positive formula over Γ) with $2^{2^m} + t + 1$ variables.*

4.4 Refining Proposition 3

Proposition 4. *Suppose R is not expressible from Γ and this is witnessed by some polymorphism of Γ of arity $k < m$. Then there exists a tiling τ of $[2^m \times 2^m]$ by \mathcal{D} with initial condition \mathbf{w} with the property that every row of τ is repeated.*

Proof. Let h be the k -ary polymorphism of Γ ; choose $\mathbf{c} = (\alpha_0, \alpha_1, \dots, \alpha_t) \in (D^k)^{t+1}$ such that $\text{proj}(\mathbf{c}) \subseteq R$ but $(h(\alpha_0), \dots, h(\alpha_t)) \notin R$. Since S is expressible from Γ , h preserves S at \mathbf{c} , so $(h(\alpha_0), \dots, h(\alpha_t)) \in S$. As $S \setminus R = \{\widehat{\top}\}$, we get $h(\alpha_i) = \top$ for all $i \leq t$.

For each $1 \leq i \leq k$ let $\mathbf{c}_i = (\alpha_0[i], \dots, \alpha_t[i]) \in R$. Define

$$\begin{aligned} M &= \{i : \mathbf{c}_i \in R_0\} \\ Q &= \{i : \mathbf{c}_i \in \{\top, \perp\}^{t+1} \setminus \{\widehat{\top}\}\} \\ Z &= \{i : \mathbf{c}_i = \widehat{\infty}\}. \end{aligned}$$

For each $i \in M$, define $\sigma(i)$ to be the unique $j \in \{0, 1, \dots, m-1\}$ such that $\mathbf{c}_i = (\beta_0[j], \dots, \beta_{t-1}[j], \gamma[j])$. Now define a map $\lambda : [2^m \times 2^m] \rightarrow D^k$ as follows: given $(x, y) \in [2^m \times 2^m]$ and $1 \leq i \leq k$,

$$\lambda(x, y)[i] = \begin{cases} \mathbf{p}(x, y)[j] & \text{if } i \in M \text{ and } \sigma(i) = j \\ \top & \text{if } i \in Q \\ \infty & \text{if } i \in Z. \end{cases}$$

We will use λ to “represent” the elements of $[2^m \times 2^m]$ as elements of D^k (though we will see below that λ is not injective). We now loosely follow the proof of Proposition 3(2) \Rightarrow (3). Suppose $(x, y) \in [2^m \times 2^m]$ and let $\mathbf{x} = \lambda(x, y)$. One can check that $\text{proj}(\mathbf{x}, \alpha_0, \dots, \alpha_t) \subseteq \mathcal{R}_{P^m \Rightarrow \Delta}$. As h is a polymorphism, this implies $(h(\mathbf{x}), h(\alpha_0), \dots, h(\alpha_t)) \in \mathcal{R}_{P^m \Rightarrow \Delta}$, i.e., $(h(\mathbf{x}), \top, \dots, \top) \in \mathcal{R}_{P^m \Rightarrow \Delta}$. Hence $h(\mathbf{x}) \in \Delta$. Thus we may define a map $\tau_h : [2^m \times 2^m] \rightarrow \Delta$ by $\tau_h[x, y] = h(\lambda(x, y))$. As in the proof of Proposition 3(2) \Rightarrow (3), it will follow that τ_h is a tiling of $[2^m \times 2^m]$ by \mathcal{D} with initial condition \mathbf{w} .

Observe that $|M| \leq k < m$, so the map σ is not surjective. Pick some $0 \leq j < m$ with $j \notin \text{range}(\sigma)$. Then the map λ has the property that if $x, x', y, y' \in \{0, 1, \dots, 2^m - 1\}$ and the binary representations of x and x' (y and y') agree everywhere except at bit j , then $\lambda(x, y) = \lambda(x', y')$. The same must therefore be true of the tiling τ_h . Hence every row (and every column) of τ_h is repeated. \square

Proposition 5. *Suppose R can be expressed from Γ by an instance of $\text{CSP}(\Gamma)$ (or primitive positive formula) with $k < 2^{2^m}$ variables. Then there exists a subset $U \subseteq [2^m \times 2^m]$ with $|U| \leq k$ such that \mathcal{D} does not tile U with initial condition \mathbf{w} .*

Proof. Choose an instance $\mathcal{P} = (X, D, \mathcal{C})$ of $\text{CSP}(\Gamma)$ and a $(t+1)$ -tuple $\mathbf{s} = (s_0, \dots, s_t)$ of variables from X such that $(\mathcal{P}, \mathbf{s})$ expresses R and $|X| = k$. Thus

$$R = \{(h(s_0), \dots, h(s_t)) : h \in \text{Sol}(\mathcal{P})\}. \quad (1)$$

For each $h \in \text{Sol}(\mathcal{P})$ define $\mathbf{c}_h = (h(s_0), \dots, h(s_t)) \in R$. Define

$$\begin{aligned} M &= \{h \in \text{Sol}(\mathcal{P}) : \mathbf{c}_h \in R_0\} \\ Q &= \{h \in \text{Sol}(\mathcal{P}) : \mathbf{c}_h \in \{\top, \perp\}^{t+1} \setminus \{\widehat{\top}\}\} \\ Z &= \{h \in \text{Sol}(\mathcal{P}) : \mathbf{c}_h = \widehat{\infty}\}. \end{aligned}$$

Next define

$$\mathcal{A} = \{x \in X : [h(x) \in P \ \forall h \in M] \ \& \ [h(x) \in \Delta \ \forall h \in Q] \ \& \ [h(x) = \infty \ \forall h \in Z]\}.$$

Similarly, define \mathcal{B} to be the set of all $x \in X$ whose values under h in M, Q, Z are in $\{0, 1\}, \{\top, \perp\}$ and $\{\infty\}$ respectively; and define \mathcal{E} to be the set of all $x \in X$ whose values under h in M, Q, Z are in $\{\mathbf{a}, \mathbf{b}\}, \{\top, \perp\}$ and $\{\infty\}$ respectively;

For each $0 \leq i < m$ choose $h_i \in M$ so that $(h_i(s_0), \dots, h_i(s_t)) = (\text{Bin}_t(i), \gamma[i])$. (Such h_i must exist by equation [1](#).) Now define $\lambda : \mathcal{A} \rightarrow P^m$ as follows: for $x \in \mathcal{A}$ and $0 \leq i < m$, put $\lambda(x)[i] = h_i(x)$.

Define $U = \{(i, j) \in [2^m \times 2^m] : \mathbf{p}(i, j) \in \text{range}(\lambda)\}$. Clearly $|U| \leq |\mathcal{A}| \leq |X| = k$. We claim that \mathcal{D} cannot tile U with initial condition \mathbf{w} . Assume to the contrary that $\tau : U \rightarrow \Delta$ is such a tiling. Define $h_\tau : X \rightarrow \Delta$ by

$$h_\tau(x) = \begin{cases} \tau[i, j] & \text{if } x \in \mathcal{A} \text{ and } \lambda(x) = \mathbf{p}(i, j) \\ \top & \text{if } x = s_j \text{ for some } 0 \leq j \leq t \\ \perp & \text{if } x \in \mathcal{B} \cup \mathcal{E} \setminus \{s_0, \dots, s_t\} \\ \infty & \text{otherwise.} \end{cases}$$

It can be shown, essentially following the proof of Proposition [3](#)($3 \Rightarrow 2$), that h_τ is a solution of \mathcal{P} . But this with the fact that $(h_\tau(s_0), \dots, h_\tau(s_t)) = \widehat{\top} \notin R$ contradicts equation [1](#). \square

5 Conclusion

Proof of Theorem [1](#). Given $n = 3m$ where $m = 2^t$, $t > 1$, take \mathcal{D}_e and \mathbf{w}_m as in Proposition [2](#)(1), and let (D, Γ_n, R_n) be the output of our construction on input $(\mathcal{D}_e, m, \mathbf{w}_m)$. (Note that D is independent of n , and $|D| = 22$ if we use the specific domino system \mathcal{D}_e described in Subsection [3.1](#).) We have $|R_n| = 3m = n$. By Proposition [3](#), R_n is expressible from Γ_n but, by Proposition [5](#), not by any CSP(Γ_n) instance having fewer than 2^m variables. \square

Proof of Theorem [2](#). Follows similarly from Propositions [2](#)(2), [3](#) and [4](#). \square

Proof sketch of Theorem [3](#). Let $\text{EXPTILE}_2(\mathcal{D})$ be the restriction of $\text{EXPTILE}(\mathcal{D})$ to inputs $(\mathcal{D}, m, \mathbf{w})$ where $m = 2^t$, $t > 1$. Standard modifications of the proof of [2](#), Theorem 6.1.2], replacing the torus with the plane as in [10](#), show that every problem $\mathcal{P} \in \mathbf{NEXPTIME}$ has a logspace reduction to $\text{EXPTILE}_2(\mathcal{D})$ for some domino system \mathcal{D} . Via a ‘‘universal domino system’’ argument we can get a single

domino system $\mathcal{D}_u = (\Delta_u, H_u, V_u)$ such that $\text{EXPTILE}_2(\mathcal{D}_u)$ is **NEXPTIME**-complete. Let $d = |\Delta_u| + 11$. Our construction and Proposition 3 give a logspace reduction of $\text{EXPTILE}_2(\mathcal{D}_u)$ to the restriction of $\neg\text{EXPR}$ to d -element domains. \square

We end with two questions.

1. Can d in Theorem 3 be reduced to $d = 3$, confirming the AIM conjecture?
2. Can Theorems 1–3 be improved so that both the domain *and* the constraint language are fixed and only the test relation varies? (Such an improvement of Theorem 3 would complement a result of Kozik for functions [16].)

References

1. Bodnarčuk, V.G., Kalužnin, L.A., Kotov, V.N., Romov, B.A.: Galois theory for Post algebras. I. Cybernetics and Systems Analysis 5, 243–252 (1969)
2. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Springer, Heidelberg (1997)
3. Bulatov, A., Krokhin, A., Jeavons, P.: Classifying the complexity of constraints using finite algebras. SIAM J. Comput. 34, 720–742 (2005)
4. ten Cate, B.: Notes on AIM CSP workshop, April 21 (2008), <http://www.aimath.org/WWN/constraintsatis/constraintsatis.pdf>
5. Cohen, D., Jeavons, P.: Tractable constraint languages. In: Dechter, R. (ed.) Constraint Processing, pp. 299–331. Elsevier, San Francisco (2003)
6. Creignou, N., Kolaitis, P., Zanuttini, B.: Structure identification of boolean relations and plain bases for co-clones. J. Comput. System Sci. 74, 1103–1115 (2008)
7. Dalmau, V.: Computational complexity of problems over generalized formulas. PhD thesis, Universitat Politècnica de Catalunya (2000)
8. Dechter, R., Pearl, J.: Structure identification in relational data. Artificial Intelligence 58, 237–270 (1992)
9. Geiger, D.: Closed systems of functions and predicates. Pacific J. Math. 27, 95–100 (1968)
10. Grädel, E.: Dominoes and the complexity of subclasses of logical theories. Ann. Pure Appl. Logic 43, 1–30 (1989)
11. Jeavons, P.: Constructing Constraints. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 2–16. Springer, Heidelberg (1998)
12. Jeavons, P.: On the algebraic structure of combinatorial problems. Theoret. Comput. Sci. 200, 185–204 (1998)
13. Jeavons, P.: Presenting constraints. In: Giese, M., Waaler, A. (eds.) TABLEAUX 2009. LNCS (LNAI), vol. 5607, pp. 1–15. Springer, Heidelberg (2009)
14. Jeavons, P., Cohen, D., Gyssens, M.: A test for tractability. In: Freuder, E.C. (ed.) CP 1996. LNCS, vol. 1118, pp. 267–281. Springer, Heidelberg (1996)
15. Jeavons, P., Cohen, D., Gyssens, M.: How to determine the expressive power of constraints. Constraints 4, 113–131 (1999)
16. Kozik, M.: A finite set of functions with an EXPTIME-complete composition problem. Theoret. Comput. Sci. 407, 330–341 (2008)

Applying Constraint Programming to Identification and Assignment of Service Professionals

Sigal Asaf¹, Haggai Eran¹, Yossi Richter¹, Daniel P. Connors²,
Donna L. Gresh², Julio Ortega³, and Michael J. Mcinnis⁴

¹ IBM Haifa Research Lab, Haifa University Campus, Haifa 31905, Israel
{sigal, haggai, richter}@il.ibm.com

² IBM Research Division, Thomas J. Watson Research Center,
P.O. Box 218, Yorktown Heights, NY, 10598
{dconnors, gresh}@us.ibm.com

³ IBM Global Business Services, 1503 LBJ Freeway, Dallas, TX 75234
julio@us.ibm.com

⁴ IBM Global Business Services, 150 Kettletown RD, Southbury, CT, 06488-2600
mike.mcinnis@us.ibm.com

Abstract. Today many companies face the challenge of matching highly-skilled professionals to high-end positions in large organizations and human deployment agencies. Non-accurate matches in these businesses can result in significant monetary losses and other negative effects. Unlike traditional Workforce Management (WM) problems such as shift scheduling, highly-skilled employees are professionally distinguishable from each other and hence non-interchangeable. Therefore, the techniques used for shift-scheduling can't be applied to the highly-skilled WM domain. Our work focuses on providing a Constraint Programming solution for supporting the assignment of highly-skilled professionals. Our experience shows that CP is well adapted to this problem. CP supports very well the underlying constraints. In addition, the rich expressive language supported by CP allows us to provide a convenient mechanism for changing and adding new matching and preference constraints. Based on this technology, we have built a tool that is currently being used by IBM service organizations and provides strong business results.

1 Introduction

Today's economy is witnessing a constantly increasing demand for skilled professionals with specialized combinations of expertise, who are essential in accomplishing high-end projects. This trend can be observed in most markets and industries. As a result, many large business organizations, as well as private and public human placement agencies, face the Workforce Management (WM) Identification and Assignment (ID&Assign) problem of assigning skilled professionals to positions with specialized requirements.

The ultimate goal therefore is to rapidly create matches that are accurate, while maximizing generated revenue. Poor decisions can result in understaffing,

under-qualification or over-qualification of assigned personnel, and high turnover of poorly matched workers. While the importance of quality matching is clear, promptly dealing with pools of hundreds of positions and professionals in a dynamic market is a serious challenge. Careful deployment of individuals is essential for boosting productivity in today's marketplace.

In spite of its importance, many companies address the WM challenge manually. Resource deployment professionals (RDPs) search for best matches based on their knowledge and their expertise, and are supported by simple query tools which provide the ability to search for a professional who is characterized by some criteria (e.g., search for a C++ developer, from New York). When the lists of professionals and positions to be matched are larger than a few dozen, this process results in assignments that are far from optimal and take a long time to create. Moreover, each RDP is usually responsible for a small number of positions and professionals and therefore takes into account only local considerations. However, a global view, comprising all positions and professionals in the pool, is essential for providing good assignments.

Given this complexity, an automated mechanism that produces a more accurate list of matches, and provides recommendations of near-optimal assignments is essential. However, developing such a system would be quite difficult both in capturing the correct abstraction level of the workforce model/rules, and in developing the underlying technology.

We applied constraint programming (CP) to develop a new tool, which successfully provides prioritization lists and near-optimal assignments. These results take into account all resources and positions in the pool, as well as the complex constraints defining a good match. As its core engine, we use a systematic CP solver developed in IBM (see [1]). The new tool was successfully piloted in 2005, and has since been essential in helping resource managers and deployment managers implement better assignments. Today it is widely deployed by IBM's Global Business Services (GBS) in all geographies.

A high-level overview of the tool was given in [2], along with many background references. In [3], we presented some technological advancements including text analysis and the flexibility feature that adjusts a given assignment when some of the parameters (e.g., professional pool, position pool or the matching constraints) have changed. In this paper, we focus on the advantages of CP as a supportive technology for a dynamic industry constraint problem. In addition, we present experimental results, including practical uses, as identified by the IBM Global Business Services organization (GBS).

Paper organization. In Section 2 we explain why we decided to use CP for this domain and provide more background information. In Section 3 we describe the WM problem in more detail, and in Section 4 we explain how we solve the problem using CP. In Section 5 we present our results. This includes experimental results on real datasets and practical uses as identified by GBS. We conclude in Section 6.

2 Why CP

Traditionally, typical WM problems address different variants of shift scheduling. In these problems, there are usually a large number of professionals, roughly divided into a small number of groups. Each group contains professionals with similar skills, and is considered to be approximately homogeneous. Professionals in the same group can be thought of (from the automation point of view) as indistinguishable and interchangeable. Given this partition to distinct groups, professionals are scheduled for shifts, where each shift requires a specific combination of personnel. Such WM problems are widely solved using traditional OR methods (e.g., linear and integer programming, reductions to other OR problems) or by other methodologies, such as modern meta-heuristics (in particular tabu search and genetic algorithms) and multi-agent systems. For example, SWOPS [4], a tool suitable for shift scheduling, is based on integer linear programming. Resource capacity planning [5] is a different WM scenario, concerned with aggregates of professionals rather than individuals. Here planning is performed to estimate future gaps and gluts in the workforce. Here too, the problem lends itself naturally to mathematical programming methods.

In contrast, the ID&Assign problem we are addressing is at the opposite extreme: the individual professionals are highly-skilled, each with his or her own unique combination of competencies, and are highly distinguishable and non-interchangeable. Additionally, it is essential to find a good match between professionals and their assigned positions; otherwise we run the risk of an under- or over-qualified assignment, or understaffing, with the obvious contingent problems. This WM problem is also inherently different from the usual supply chain problems in OR. Our entities are people rather than parts; we cannot model them as pure sets of attributes. Individuals have their own unique skills, behaviors, interests, and expectations.

This ID&Assign problem has not been addressed before in the literature and seems to be harder to automate. The traditional OR methods listed above generally fail on this problem for a number of reasons. First, the constraints, which depend on the particulars of professionals and positions, are complex and do not translate easily to linear constraints. This as opposed to the simple constraints, such as vacation-time and maximum daily work hours, seen in mainstream workforce scheduling applications. Second, most OR methods rely on optimizing an objective function. In our case, it is nearly impossible to put a price tag on most of the variables involved. For example, how can we quantify the cost of a dissatisfied customer or a displeased professional resulting from a non-perfect match? Finally, new rules and constraints arise frequently. To handle them quickly and efficiently, the desired mechanism should have a rich expressive language that will easily allow the formulation and maintenance of these constraints. Translating the problem into a linear model would create a maintenance nightmare as the model would be very far from the original constraints.

Our tool relies on Constraint Programming (CP) methodology. The expressive language of CP methodology is rich, natural, and modular, with many types of constraints, therefore allowing the rapid development and maintenance of

models. Additionally, the strong algorithmic foundation of CP allows for fast execution and good optimality. Therefore, it suits the nature of our WM problem better than traditional OR methods. In the past, there have been a few attempts to employ CP in solving WM problems, although these cases were scheduling problems of a more traditional nature (e.g., British Telecom used CP to solve a real-life problem [6] that was later also solved in [7]. J. Metivier et al. solved the nurse rostering problem in [8]).

3 Problem Definition

The tool's main inputs consist of lists of available professionals and open positions, a set of matching rules, and a set of prioritization rules. These inputs are dynamic in that the characteristics of the professionals and open positions, as well as the matching and prioritization rules, are changed on a regular basis. The problem characteristics can be different between different organizations, and even between different areas of the same organization. Moreover, the problem definition may be changed over time by the organization's administration or by an RDP exploring different possibilities. For example, the RDP may want to check for a potential assignment given that the maximal allowed distance between the professional's location and the position's location is within 50 km, and later check how the assignment differ given a maximal distance of 100 km.

In the following subsections, we describe in more detail the problem's main inputs, while concentrating on its dynamic nature.

3.1 Position and Professional Definition

Each professional is characterized by a set of attributes, such as availability dates, set of skills, and location. In a similar way, each position is characterized by a set of attributes, including start date and position duration. These sets of attributes may be different for different problems.

A modeling interface should provide the ability to define these attributes. An attribute definition includes a name and its type. The type can be a basic type such as string, integer, and date, or a set of elements all from the same basic type. For example, the attribute *education* for a professional may include the list of courses the professional participated in.

3.2 Matching Rules

The matching rules can be of two types:

- *Built-in matching rules* — Basic set of matching rules such as availability rule, location rule, or skill matching rule. These rules are controlled through a set of parameters. For example, a possible integer parameter for the location rule is “The maximum distance between the professional's location and the job location”. For the skill matching rule, we will find a boolean parameter such as “Should we consider the professional's secondary skills?”

- *Specific matching rules*— Each matching problem may include specific matching rules. For example, one may want to add a matching rule that considers years of experience. CP naturally supports this type of matching rule, by simply adding a new constraint using CP language, such as [9] and [10]. For example, consider the experience years matching rule as follow:

$$\begin{aligned} &(\textit{position.exp_level} = \textit{beginner}) \rightarrow (\textit{person.exp_years} < 2) \textit{ and} \\ &(\textit{position.exp_level} = \textit{expert}) \rightarrow (\textit{person.exp_years} > 6) \textit{ and} \\ &(\textit{position.exp_level} = \textit{professional}) \rightarrow ((\textit{person.exp_years} \geq 2) \textit{ and} \\ &(\textit{person.exp_years} \leq 6)) \end{aligned}$$

Another type of specific matching rules are those that correspond to a particular position or professional, as shown in the example below.

$$(\textit{position.identification} = A1) \rightarrow (\textit{person.degree} \in \{MA, PHD\})$$

3.3 Prioritization Scheme

As stated above, assigning the right professional to the right position requires careful consideration. While the matching rules represent a threshold that needs to be passed, the prioritization rules ensure high-quality solutions.

The prioritization mechanism ranks all professionals who match a specific position, and ranks all positions that match a specific professional. This ranking is considered later in the CP problem, when searching for the best solution.

The prioritization scheme contains an ordered list of prioritization rules such as “prefer a professional who lives closer to the position.” The final ranking of each pair is based on an aggregate calculation of these rules. Since it is nearly impossible to put a price tag on most of the variables involved, the RDPs define their desired priority hierarchy by picking an ordered list of relevant priority rules. Given two possible matches, comparison is done according to the order defined by the user, i.e., the match that satisfies a higher priority rule wins (i.e., lexicographic order).

The prioritization scheme is dynamic. In each run, the RDPs may want to explore different prioritization scheme by exploring different order of the rules, different parameters for the rules or even adding special rules that are specific to the current WM problem. A flexible modeling mechanism based on CP constructs naturally supports such an environment.

We propose two types of prioritization rules:

- *Built-in prioritization rules*. For example, “prefer a professional who lives closer to the position location”.
- *User-defined prioritization rules*. For this, we expose the underlying optimization language and provide constructs such as “minimize”, “maximize”, and “order”. A declarative language for the prioritization rules provides the ability to state expressions such as:

- $\text{minimize}(|(\text{position.required_exp_years} - \text{person.exp_years})|)$ — prefers a match with a smaller gap between the position’s required years of experience and the professional’s years of experience.
- $\text{match}(\text{City})$ — prefers a position and professional that are located in the same city.
- $\text{order}(\text{person.employment_status}(\text{RG}, \text{PT}, \text{SC}))$ — prefers a regular professional over a part-time professional over a sub-contractor professional

4 Modeling WM ID and Assign Using CP

There are two major types of WM challenges need to be considered.

- *Complete Assignment* — construct an explicit near-optimal assignment, attempting to staff as many positions as possible, while enforcing a no-overlap constraint – the same professional cannot be assigned to positions that overlap in time.
- *Feasibility solution* — Provide a list of feasible professionals for each position and a list of feasible positions for each professional. The order of the professionals/positions in the feasible lists follows the prioritization scheme and may also consider global considerations such as the number of feasible matches per each position and per each professional.

In this paper, we present in detail the CP solution for the Assignment problem. For the Feasibility problem, we use the same modeling as used by the Assignment problem. However, in the Feasibility problem, the solver stops after it reaches the first arc-consistency state (For more information, see [11])

4.1 The *no-overlap* Requirement

Our goal is to maximize the number of positions assigned, while maintaining the best fit of professionals to their assigned positions. The basic constraint we wish to enforce is that while the same professional can be assigned to multiple positions, these positions cannot have overlapping execution times.

We model each position by a variable whose initial domain is the entire set of professionals. As part of the first arc-consistency the domain is reduced to include only the professionals who are qualified to perform it and are available throughout its duration. Suppose every pair of positions overlapped in their execution time. In that case, we could define a single *alldifferent* constraint over all variables, thereby guaranteeing that no person is assigned to two positions. Since, in general, not all positions overlap (for example, position *A* may end in June, while position *B* starts in August), we employ the *somedifferent* constraint [12] instead. The *somedifferent* constraint is a natural generalization of *alldifferent* that answers our needs. It is defined over a subset of the variables, together with an underlying graph whose vertices are the participating variables. The constraint requires that variables that are adjacent in the graph are assigned

different values. (Note that the *alldifferent* constraint is the special case obtained when the underlying graph is complete.) Formulated mathematically,

$$\text{somedifferent}_G(v_1, \dots, v_k) = \{(a_1, \dots, a_k) : a_i \in D_i, (v_i, v_j) \in E(G) \rightarrow a_i \neq a_j\},$$

where $E(G)$ is the set of edges of the graph G , and D_i the domain of variable v_i . We note that defining a single constraint, rather than a separate constraint for each pair of conflicting variables, guarantees better pruning during propagation. The only concern with *somedifferent* is that its propagation is an NP-hard problem (since, e.g., it generalizes coloring problems), and therefore it is not likely that an efficient (polynomial) propagator exists. However, there exists a non-trivial propagator (see [12]), which together with a few heuristics, works well in practice. Still, defining this constraint over a large set of variables is not recommended.

4.2 The CSP Model

Variables and domains. Each position is modeled by a variable whose initial domain is the entire set of professionals. As part of the first arc-consistency, the domain is reduced to include only the professionals who are qualified to perform it and are available throughout its duration. We define a single fictitious value which we add to all domains. We treat this value specially: although we admit it as viable in the instantiation phases, we ignore it in the propagation phases, in effect allowing it to be assigned to overlapping positions. The reason we introduce this fictitious value is because it is quite likely we will not be able to staff all positions due to insufficient professionals. Ordinarily, in such cases we would simply get an indication that the problem is unsatisfiable. By adding the fictitious value, we can guarantee solvability, and by using value ordering, we can direct the solver to prefer real professionals over the fictitious one. Of course, once we obtain a solution, we remove the fictitious value and reject all positions to which it has been assigned.

Hard constraints. The hard constraints should guarantee that the assigned professionals match the positions, and that no professional is assigned to two positions that overlap in time.

We guarantee that the professionals match the positions by adding all the matching rules to the CP model.

The no-overlap requirement can be accomplished by using a single *somedifferent* constraint whose underlying graph contains an edge between every two overlapping positions. However, because the propagator for *somedifferent* cannot be efficient (i.e., cannot run in worst-case polynomial time), we use the following partitioning heuristic, which results in several *somedifferent* constraints, each applying to a small underlying graph.

- Edges in the full *somedifferent* graph connecting pairs of variables with disjoint domains are obviously redundant. We delete them. We then partition the resultant graph into its connected components.

- We partition each connected component into clusters of size t (a user-defined threshold set by default to 10). If the size of the connected component is not divisible by t , one of the clusters will be smaller than t . We apply a *somedifferent* constraint to each cluster.
- We further add an approximate *somedifferent* constraint over each connected component that is larger than t . The approximate constraint has the same semantics as the ordinary one, but is associated with an efficient (polynomial) propagator (see [12]). The drawback is that this propagator may not filter all unsupported domain values.

When using an approximate some-different propagator, we risk creating sub-optimal staffing: the approximate propagator may result in sub-optimal pruning of the search tree at the arc-consistency stage, possibly leading to collisions on the same position, which may have been avoided had the propagation been exact. In practice, our analysis shows that the suggested heuristic described above works well and achieves a considerable speedup compared with both a single *somedifferent* constraint and a single approximate *somedifferent* constraint, applied to all variables.

Value ordering. We apply two types of value ordering. First, we prefer the assignment of real versus fictitious persons. Second, we apply user preferences, which are typically hard to quantify. We use the preference scheme as defined in Section 3.3 to sort all the professionals who match a position. For each match, we attach a match-quality object. The match-quality objects contain a match quality indicator for each preference rule (e.g., the indicator for the location preference rule describes the distance between the professional location and the position location). Then we sort all the match quality objects that corresponds to a specific position. The sort order follows the lexicographic order of the preference rules as defined in the preference scheme. Finally, we choose the professionals whose attached match-quality is located in the highest rank.

Support dynamic model. As stated in Section 3, the matching rules and the prioritization scheme are changed frequently. They may vary between different organizations and between different runs of the same organization.

CP naturally supports this type of problem through declarative constraint language such as MiniZinc [9] and the Optimization Programming Language (OPL [10]). The constraint language provides a natural way for defining the matching rules. Optimization constructs such as minimize and maximize provide the additional layer for supporting the prioritization rules.

To support such a dynamic model, we expose the corresponding part of the underlying CP language to the user. The built-in matching constraints are updated by setting their corresponding parameters, and both the built-in matching constraints and the user-defined matching constraints are added to the problem.

The preference rules define the value ordering of each position variable. The value ordering is based on the match-quality objects as have been described in the ‘value ordering’ subsection. Each match-quality objects include an indicator for each built-in preference rule and for each user defined preference rule.

5 Experiments and Practical Usage Discussion

The methods we described were integrated in IBM workforce management tool, and provide new capabilities beyond traditional database queries (e.g., search for all professionals with band 8). The solutions for the *Feasibility problem* help both the RDPs and the professionals to identify potential matches, and the solutions for the *Assignment problem* are mainly used for capacity planning.

In this section, we first present an analysis of the Assignment mode, including running time and quality analysis. We then present in more detail the practical usage scenarios of both the *Feasibility problem* and the *Assignment problem*.

5.1 Analysis of Experiments

We experimented with two real datasets from GBS. These were run on an Intel(R) Xeon(TM) 2.4 GHz machine with 2.5 GB of RAM. The tests, described in detail in Table 1, included hundreds of available positions and thousands of professionals. The last column presents the maximum number of positions that are active on any given date. This provides some indication of the difficulty of the assignment problem.

Table 1. Experiments: Details of the experimental datasets

Input	Input description	Number of positions	Number of people	Max overlapping positions
AP	Professionals and project positions in the Asia Pacific region (i.e, China and other countries in Southeast Asia)	464	6232	351
USAMS	Professionals and project positions targeted for the US Application Management Services organization	622	4882	195

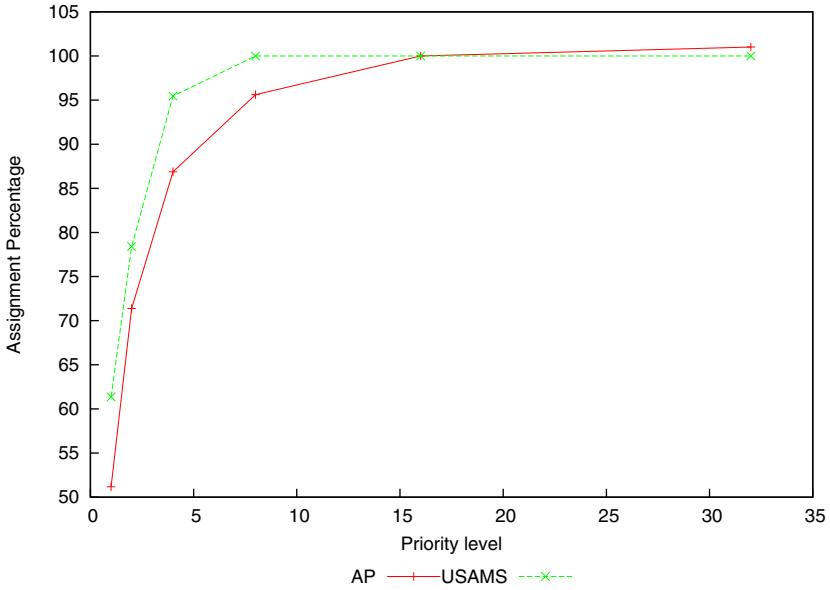
Table 2 shows the results of generating the feasibility solutions on the two datasets. The table presents the number of positions that one or more professionals match, and the maximum and the average (avg) number of professionals that match a position. The same type of data is also presented for professionals. The last column presents the number of different priorities each position has (e.g., two professionals that match the same position may have equal match quality, or may have different priorities and therefore we would prefer one of them over the other). The number of priorities for each position demonstrates how well the priority rules used differentiate between the candidates. The results show that the number of priorities was close to the number of matching candidates.

Table 2. Results of the prioritization mode experiments

Input	Time (sec)	Num of positions matched	Num of people matched	Positions per person (max/avg)	People per position (max/avg)	Priorities per position (max/avg)
AP	21	351	862	20 / 0.46	77 / 8.13	58 / 8.71
USAMS	20	202	101	26 / 0.10	16 / 0.91	11 / 2.59

Table 3. Results of the assignment mode experiments

Input	Time (sec)	Number of assignments	Percent assigned	Assignment priority (max/avg)
AP	80	300	85%	19 / 2.69
USAMS	115	88	44%	7 / 1.77

**Fig. 1.** Distribution of the assignment priorities

Finally, Table 3 shows the results of the assignment mode run. The average priority of an assignment provide an indication of the quality of the assignment. In Figure 1, the distribution of the priorities is shown in more depth. The figure shows that most positions are assigned their preferred first or second priority professionals.

5.2 Prioritization Mode Practical Usages

IBM's Global Business Services (GBS) unit provides project-based professional services to its customers through its workforce of over 100,000 consultants and specialists worldwide. Assigning the correct mix of specific skills and expertise levels is critical to the success of projects, both from the point of view of meeting customer needs and from that of meeting IBM's profit objectives.

GBS has been applying the new technology to address the problem of matching professionals to project requirements. Analysis of the data required for both professionals and positions as well as developing the right set of matching and prioritization rules were done in conjunction with GBS Business leaders to define possible ways of meeting project scheduling requirements, professional availability, skills requirements, as well as to give higher priority to matches that will result in improved profit performance.

Project managers and resource deployment professionals already have at their disposal a number of search tools they can use to find professionals to fill project positions. Similarly, professionals can search for projects that match their qualifications. However, the number of matches returned by a search can be daunting. The new technology can find a more manageable set of high-quality matches based on a relatively large number of pre-defined requirements.

The data available for the matching and the set of matching and prioritization rules are key to generating good assignment suggestions. IBM has developed databases and processes to support a collection of rich and high-quality data about the professionals and positions. On top of this data, we developed the corresponding matching and prioritization rules. The data for each professional includes a set of qualifications, location, a list of languages the professional is familiar with and at which level, the professional's preference in terms of work location (e.g., overseas positions), availability start-date and availability indicators (e.g., a professional who reports less than 10 working hours a week can be considered as available), and more. The data for each position includes the position's start and end dates, the required skill, the job location, whether the position can be done remotely, which languages are required and at what level, and more.

Matching skills is a key rule. IBM has developed a corporate skills taxonomy with several thousand sets of skills that may be required by a position. A professional is assigned a primary skill plus a set of secondary skills, and a set of inferred skills. The last one is created automatically by analyzing the professional data. (For example, the system may infer that a C++ developer is also qualified for C coding). In simple skill searches, a professional may only be considered a candidate for a position if his/her primary skill is the same as the skill requested for the position. Our method allows the skill to be matched based on multiple methods, including the secondary skills, the inferred skills, or by comparing the degree of match between the textual description of a position requirements and a professional's resume. Thus, the new technology can often find positions that were not considered in simpler skill searches.

In addition, the tool allows for the specification of rules to match on multiple languages required by the position, professional travel preferences, whether the work can be done remotely or whether the customer will pay for travel, whether the professional meets citizenship and security clearance requirements, and more. This helps reduce the search to a reasonably small number of matches that can then be more easily examined by RDPs or the interested professionals.

GBS has been employing the new technology in two scenarios: 1) Supporting Project Managers and Resource Deployment Professionals in their daily activity of fulfilling project requirements or finding work for professionals with time available to work on new projects, and 2) Supporting professionals in their search for projects suitable to their skills.

In the first scenario, a daily report is published for each GBS geographic organization listing all project positions and a small number of feasible candidates for each position. Project managers can then choose to further investigate a professional's qualifications and propose them for a project. Since finding the right professional for each project position is essential to the success of the project, personal interviews of a small group of candidates is the norm before making a final assignment. Similarly, a report is generated listing professionals that each RDP is responsible for, along with matching positions for each one of them. RDPs can then work with these professionals, propose suitable positions, and work to help them make it to the short list for personal interviews.

In the second scenario, the objective was to provide actionable information to the people that have the strongest motivation for finding a project position, i.e., professionals that have completed or will soon complete their involvement in a project and are thus in need of finding a new project assignment. The matching results of the new technology are used to create a tailored set of suitable project positions for these professionals. This list is sent directly to the professionals via email on a weekly basis. If they are interested, the professionals can then obtain more information about these suitable positions and contact their RDPs or market themselves to the positions' project managers, thus giving them more control of their own future.

5.3 Assignment Mode Practical Usages

One may ask why we do not simply provide the optimal set of assignments, but rather provide a number of feasible matches. When we built the initial tool version, we focused on the assignment problem, thinking that the users of the tool would obviously want an assignment that maximized the number of assigned positions, while maintaining high quality matches. However, during early testing of the tool, early users of the tool reported dissatisfaction with proposed assignments. Although the assignments generated by the tool adhered to the mandatory matching rules and prioritization schemes, the RDPs often had additional information about the open seats or the professionals they would use to invalidate a proposed match or suggest what they believed to be better match. For example, the RDP may know that the proposed candidate does not get along with the project manager of the project corresponding to the open seat.

An RDP may want to assign a different professional to an open seat because the RDP believes the assignment would be a good job-growth exercise for that other professional. The RDPs felt that the optimized assignment was too rigid for them. They could easily poke holes in the assignment using their information and objectives. We recognized that in this space of workforce management, the new decision support tool we developed needed to be more flexible and so we created the feasibility mode as an alternative method of producing assignments.

Still, there is an important practical scenario which makes full use of the assignment problem. Resource managers and deployment managers are very concerned about overdue open seats, that is, open seat positions that should have already started work but are unfilled, and open seats that are due to start in the near-future, for example, the next thirty days. For open seats beyond thirty days, the managers usually have enough time to consider other sources of supply, such as contractors, or they can upskill or train an existing professional to perform the work. Typically for overdue open seats and open seats due to start in the next month or so, contracts have been signed or the work has been committed to, so the professional and deployment managers must try to fill those open seats. To help them determine where they will have problems filling these open seats, we run the new technology in the assignment mode. We consider the open seats that are overdue and those that are due to start in the next thirty days. We consider the professionals that are currently available and those professionals that will become available in the next thirty days. After running the tool in the assignment mode, we then report the open seats that were not matched in the assignment, the *missed assignments*. The missed assignment report gives the resource and deployment managers a good idea of how many assignments they will be able to fill and which assignments will be troublesome to fill in the near-future.

6 Concluding Remarks

We described a CP approach to the ID&Assign problem of highly-skilled professionals. CP has many advantages over traditional OR methods in solving this problem, most notably its separation between problem modeling and algorithmic foundations, which enables easy modeling of complex rules, and rapid adjustment to newly created constraints. The tool we developed demonstrates the applicability of CP to the problem, and shows that large industrial-scale problems can be solved with near-optimal results and with real-time performance. It is aimed at automating the tedious and repetitive tasks performed manually by resource deployment managers, while allowing them to concentrate on real decision-making. As such, our main direction of current development is in modeling and solving complex CSPs that arise when building coherent teams of professionals for assignment to large projects.

Acknowledgments. We would like to thanks Steve Heise, Crystal Howell, and Will Riddle from GBS for their contribution in applying the technology in a business environment.

References

1. Naveh, Y., Rimon, M., Jaeger, I., Katz, Y., Vinov, M., Marcus, E., Shurek, G.: Constraint-based random stimuli generation for hardware verification. *AI Magazine* 28, 13–30 (2007)
2. Naveh, Y., Richter, Y., Altshuler, Y., Gresh, D.L., Connors, D.P.: Workforce optimization: Identification and assignment of professional workers using constraint programming. *IBM J. Res. and Dev.* 51(3), 263–279 (2007)
3. Richter, Y., Naveh, Y., Gresh, D.L., Connors, D.P.: Optimatch: Applying constraint programming to workforce management of highly-skilled employees. In: *IEEE/INFORMS International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pp. 173–178 (2007)
4. Gilat, D., Landau, A., Ribak, A., Shiloach, Y., Wasserkrug, S.: Swops– shift work optimized planning and scheduling. In: *Proc. 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pp. 518–523 (2006)
5. Gresh, D.L., Connors, D.P., Fasano, J.P., Wittrock, R.: Applying supply chain optimization techniques to workforce planning problems. *IBM J. Res. and Dev.* 51(3), 251–261 (2007)
6. Munaf, D., Tester, B.: And/or parallel programming in practice. Technical Report WP12:1203, British Telecom Research Lab, London, UK (1993)
7. Yang, R.: Solving a workforce management problem with constraint programming. In: *The 2nd International Conference on the Practical Application of Constraint Technology*, pp. 373–387 (1996)
8. Metivier, J., Boizumault, P., Loudni, S.: Solving nurse rostering problems using soft global constraints. In: Gent, I.P. (ed.) *CP 2009. LNCS*, vol. 5732, pp. 73–87. Springer, Heidelberg (2009)
9. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: *Minizinc: Towards a standard cp modelling language*. In: Bessière, C. (ed.) *CP 2007. LNCS*, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
10. van Hentenryck, P.: *The OPL optimization programming language*, Cambridge, MA, USA (1999)
11. Dechter, R.: *Constraint Processing*. Elsevier Science, Amsterdam (2003)
12. Richter, Y., Freund, A., Naveh, Y.: Generalizing alldifferent: The somedifferent constraint. In: Benhamou, F. (ed.) *CP 2006. LNCS*, vol. 4204, pp. 468–483. Springer, Heidelberg (2006)

Computing the Density of States of Boolean Formulas

Stefano Ermon, Carla P. Gomes, and Bart Selman

Cornell University, Ithaca NY 14850, USA
{ermonste,gomes,selman}@cs.cornell.edu

Abstract. In this paper we consider the problem of computing the density of states of a Boolean formula in CNF, a generalization of both MAX-SAT and model counting. Given a Boolean formula F , its density of states counts the number of configurations that violate exactly E clauses, for all values of E . We propose a novel Markov Chain Monte Carlo algorithm based on flat histogram methods that, despite the hardness of the problem, converges quickly to a very accurate solution. Using this method, we show the first known results on the density of states of several widely used formulas and we provide novel insights about the behavior of random 3-SAT formulas around the phase transition.

1 Introduction

Boolean satisfiability (SAT) solvers have been successfully applied to a wide range of problems, ranging from automated planning to hardware and software verification. In all these applications, the original problem is encoded into a Boolean formula and the task is that of deciding whether it is satisfiable or not.

Given the tremendous success of SAT solvers, a lot of attention has been directed toward extending this technology to the model counting problem ([1,2,3]), that is the problem of computing the number of distinct satisfying assignments for a given propositional formula. This task is also very important because of its wide range of applications. For example, several probabilistic inference problems in graphical models such as Bayesian inference can be effectively translated into model counting ([4,5]). Moreover, when a SAT encoding is used to solve hard combinatorial problems arising in other domains, knowledge of the number of solutions can usually provide useful insights into the original problem.

Another very active line of research is devoted to the study of the optimization version of SAT, namely the maximum satisfiability problem (MAX-SAT). In MAX-SAT the goal is to find a truth assignment that satisfies the maximum possible number of clauses of a given Boolean formula in conjunctive normal form (CNF). This problem is important because many fundamental graph theoretic problem such as MAX-CUT, MAX-CLIQUE, Minimum Vertex Cover have linear time encodings as MAX-SAT. Moreover MAX-SAT has direct applications in a wide range of domains such as routing problems and expert-systems (see e.g. [6]).

In this paper we consider the problem of computing the density of states of a Boolean formula in CNF, which is a generalization of both MAX-SAT and model

counting. The density of states (DOS) counts the number of truth assignments or configurations that violate exactly E clauses, for all values of E . In other words, the problem is to compute the number $n(E)$ of configurations that leave exactly E clauses unsatisfied, for all possible values of E . The density of states is a very detailed characterization of the configuration space associated to a formula. In particular, $n(0)$ is the number of satisfying assignments or models of the formula. The lowest value of E with a non-zero density (i.e. $\min_E \{E | n(E) > 0\}$) is the solution of the corresponding MAX-SAT problem.

Given that computing $n(0)$ is equivalent to model counting, the problem of computing the entire density of states is at least as hard as model counting and therefore it is $\#P$ -hard.

The name density of states is borrowed from statistical and condensed matter physics, where the density of states (DOS) of a system describes the number of states at each energy level that are available to be occupied. For SAT instances, we simply define the energy $E(\sigma)$ of a configuration σ to be the number of clauses that are not satisfied by σ . In physics the density of states represents a deep characterization of the system, that is used to investigate various physical properties of matter and to explain a number of quantum mechanical phenomena. Analogously, in SAT the density of states gives a fine characterization of the search space which can provide further insights into the development of new algorithms.

We propose **MCMC-FlatSat**, a novel Markov Chain Monte Carlo sampling technique to estimate the DOS for Boolean formulas, that is inspired by recent methods introduced to estimate the DOS for statistical physics models [7]. Our technique outperforms standard Metropolis sampling by overcoming the often impractical mixing times. Moreover our method is especially suitable to deal with rough energy landscapes with multiple local minima in the free energy that are typical of combinatorial problems.

We empirically demonstrate that **MCMC-FlatSat** converges quickly to a very accurate solution. Using this new method, we obtain novel insights about the behavior of random 3-SAT formulas around the phase transition. Moreover, we are able to show the first known results on the shape of the density of states for several widely used formulas from the SATLib benchmark. Our results are very promising and we expect that this new approach will find many other applications both to counting and inference problems.

2 Density of States: Problem Definition

In this paper we consider the problem of computing the density of states of a given Boolean formula F in conjunctive normal form (CNF). A *clause* C is a logical disjunction of a set of (possibly negated) variables. A formula F is said to be in CNF form if it is a logical conjunction of a set of clauses \mathcal{C} .

We define V to be the set of propositional variables in the formula, where $|V| = n$. A variable assignment $\sigma : V \rightarrow \{0, 1\}$ is a function that assigns a value in $\{0, 1\}$ to each variable in V . As usual, the value 0 is interpreted as FALSE and

the value 1 as TRUE. A variable assignment σ will also be interchangeably called a *configuration*, a term that refers to an element of $\{0, 1\}^n$, a set isomorphic to the set of all possible variable assignments.

Let F be a formula in CNF over the set V of variables with $m = |C|$ clauses and let σ be a variable assignment. We say that σ satisfies a clause C if at least one signed variable of C is TRUE. We define the *energy* of a configuration $E(\sigma)$ to be the number of clauses that are unsatisfied when F is evaluated under σ . If $E(\sigma) = 0$, then σ satisfies F and σ is called a model, solution, a ground state or satisfying assignment for F .

Given a Boolean formula F , the *density of states* (DOS) $n(\cdot)$ is the function $n : [0, \dots, m] \rightarrow \mathbb{N}$ that maps energy levels to the number of configurations with that energy level:

$$E \mapsto |\{\sigma \in \{0, 1\}^n | E(\sigma) = E\}|.$$

It is clear from the definition that the DOS of any formula F satisfies the normalization constraint $\sum_{E=0}^m n(E) = 2^n$.

3 Prior Work

Despite the rich literature devoted to the study of model counting and MAX-SAT, there is little prior work on the more general problem of the computation of the density of states.

In [8] the authors propose sampling uniformly at random N configurations $\sigma_1, \dots, \sigma_N$ and then estimating the DOS with an energy histogram $h(E)$ based on the samples. This approach is clearly unpractical because it requires an enormous number of samples to get an accurate description of the DOS. In particular, any attempt to sample at least a constant fraction of the whole configuration space is doomed to have exponential complexity.

A more sophisticated sampling scheme is proposed in [9]. The authors propose the use of a Monte Carlo simulation with standard Metropolis transition probabilities between configurations σ_i and σ_j given by $p_{i \rightarrow j} = \min\{1, e^{\frac{E_i - E_j}{T}}\}$, where E_j is the number of unsatisfied clauses by σ_j and T is a *temperature* parameter. Upon convergence, it is well known that the steady state distribution $P(i)$ is Boltzmann distributed with the correct energy function $E(\cdot)$ (that measures the number of unsatisfied clauses). The density of states can then be obtained from the *canonical ensemble* rule $n(E) = P(E)e^{-\frac{E}{T}}$. It is well known that the Metropolis algorithm can have very slow mixing times, especially when dealing with rough energy landscapes with multiple local minima in the free energy ([7][10]). Unfortunately combinatorial energy landscapes, such as the one corresponding to the energy used here, are known to have many free energy minima and a similar problem of long tunneling times between local minima arises. These reasons intuitively explain why the use of the Metropolis algorithm is unpractical to deal with Boolean formulas. In the experiments we conducted, we observed convergence only on very small instances and only for certain temperature ranges.

4 A Novel Sampling Strategy: The Flat Histogram Method

We propose a Markov Chain Monte Carlo method to compute the density of states based on the flat histogram idea that is inspired by recent work developed by the statistical physics community [7] to avoid Metropolis sampling. The central idea of this method is that if we perform a random walk in the configuration space $\{0, 1\}^n$ such that the probability of visiting a given energy level E is inversely proportional to the density of states $n(E)$, then a flat histogram is generated for the energy distribution of the states visited. Suppose we define a random walk with the following transition probability

$$p_{\sigma \rightarrow \sigma'} = \min \left\{ 1, \frac{n(E)}{n(E')} \right\} \quad (1)$$

of going from a configuration σ with energy E to a configuration σ' with energy E' . The detailed balance equation

$$P(\sigma)p_{E \rightarrow E'} = P(\sigma')p_{E' \rightarrow E}$$

is satisfied when $P(\sigma) \propto 1/n(E)$. This leads to a flat histogram of the energies of the states visited because $P(E) = \sum_{\sigma: E(\sigma)=E} P(\sigma) = \text{const.}$

Since the density of states is unknown a priori, and computing it is precisely the goal of the algorithm, it is not possible to construct a random walk with transition probability (1). However it is possible to start from an initial guess of the DOS and keep changing the current estimate $g(E)$ in a systematic way to produce a flat energy histogram and simultaneously make the density of states converge to the true value $n(E)$.

MCMC-FLATSAT(ϕ)

- 1 Start with $g(E) = 1$ for all E
- 2 Start with a modification factor $F = F_0$
- 3 **repeat**
- 4 **repeat**
- 5 Generate a new state and accept with prob. given by eq. (1)
- 6 Adjust $g(E) : g(E) = g(E) \times F$
- 7 Increase visit histogram $H(E) \leftarrow H(E) + 1$
- 8 **until** until H is flat
- 9 Reduce F
- 10 Reset the visit histogram H
- 11 **until** F is close enough to 1
- 12 Normalize g
- 13 **return** g

To generate a new configuration we randomly flip a variable with uniform probability, but other strategies are possible as well. The modification factor F plays a critical role because it controls the tradeoff between the convergence rate

of the algorithm and its accuracy. Large initial values of F imply a substantial diffusion rate and therefore fast convergence to a rather inaccurate solution. This rough initial estimate is subsequently refined as the value of F decreases until $F \approx 1$, at which point when a flat histogram is produced $g(E)$ has converged to the true density $n(E)$.

Due to statistical fluctuations, a perfectly flat histogram occurs with an extremely low probability. Therefore in our implementation we use a flatness parameter; in our experiments it is set so that an histogram is considered flat when all the values are between 90% and 100% of the maximum value. The value of F is reduced according to the schedule $F \leftarrow \sqrt{F}$, with an initial value $F_0 = 1.5$; the impact of the schedule on the convergence rate is an open research question. By construction the DOS is obtained only up to constant factors: the normalization of g ensures that $\sum_E g(E) = 2^n$, where n is the number of variables in the formula.

5 Effectiveness and Validation of MCMC-FlatSat

The goal of this section is to verify the convergence of MCMC-FlatSat and to empirically evaluate the accuracy of the solution obtained. To accomplish these results, we first empirically check the accuracy of the results obtained for small structured formulas, for which we can compute the true density by exact enumeration of the entire (exponentially large) state space. We also test MCMC-FlatSat on larger synthetic formulas for which we derive an analytical expression for the true density of states, as well as on random 3-SAT formulas. For larger structured instances, for which no known method can be used to compute the true DOS, we make use of partial consistency checks to validate the results.

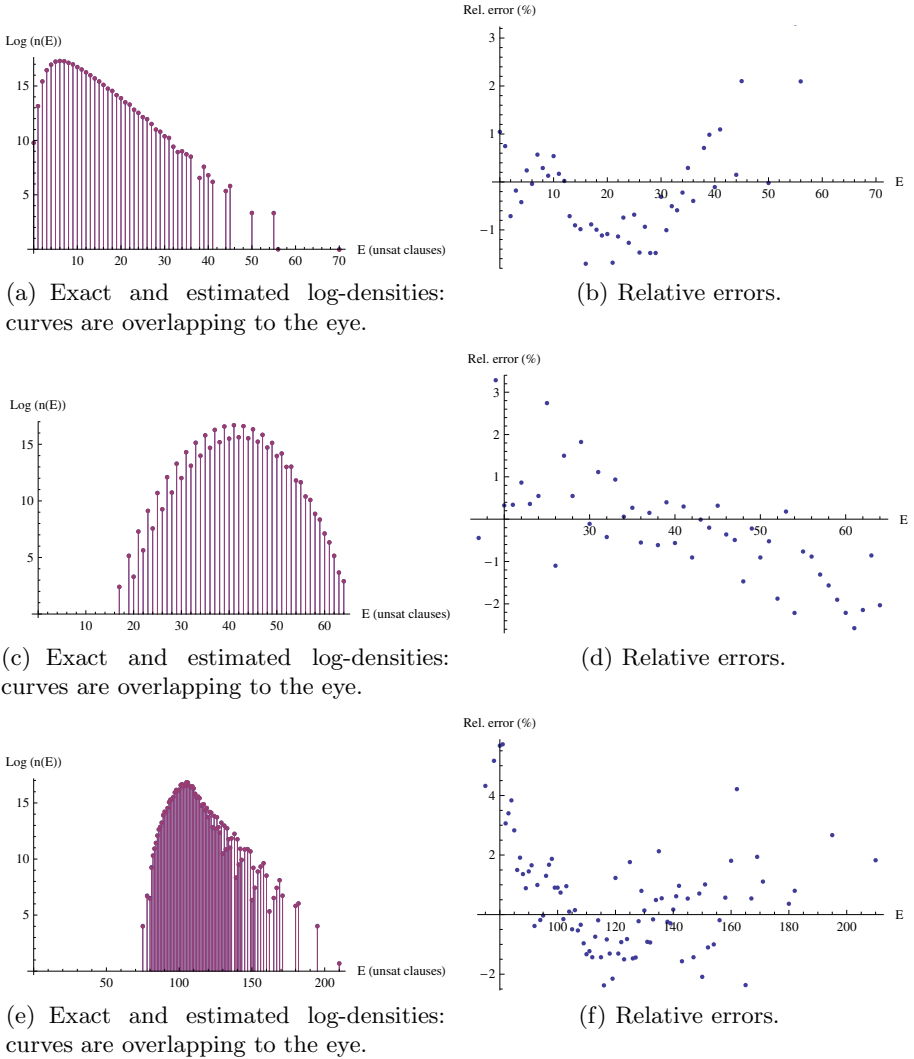
When the true DOS is known, we employ two metrics to evaluate the accuracy of the results: the relative error for each data point and a global measure represented by the Kullback-Leibler divergence between the true and the estimated densities. The Kullback-Leibler divergence between the true density $n(\cdot)$ and the estimated one $g(\cdot)$ is defined as:

$$D_{KL}(n||g) = \sum_{E=0}^m \frac{n(E)}{Z} \log \left(\frac{n(E)}{g(E)} \right)$$

where $Z = 2^n$ is used to normalize the DOS to probability distributions. In fact, the KL divergence is a standard information theoretic non-symmetric measure of the difference between two probability distributions P and Q . In information theoretic terms, the KL divergence measures the expected number of extra bits required to code samples from P when using a code based on Q , rather than using a code based on P .

5.1 Structured Problems: Exact Counts

In figure [1](#), we compare the true and estimated log-densities for several small instances (all with less than 28 variables) from the MAXSAT-2007 competition



(a) Exact and estimated log-densities: curves are overlapping to the eye.

(b) Relative errors.

(c) Exact and estimated log-densities: curves are overlapping to the eye.

(d) Relative errors.

(e) Exact and estimated log-densities: curves are overlapping to the eye.

(f) Relative errors.

Fig. 1. The density of states of a Boolean formula counts the number of configurations that violate exactly E clauses, for all values of E . We present a comparison of the estimated density ($g(E)$) and the exact one ($n(E)$) computed by explicit enumeration for several small instances from the MaxSAT-2007 benchmark. Figures [1\(a\)](#), [1\(b\)](#) are relative to the Ramsey Theorem instance *ram_k3_n8.ra0.cnf* (28 variables, 126 clauses). Figures [1\(c\)](#), [1\(d\)](#) are relative to the Spin Glass instance *t3pm3-5555.spn.cnf* (27 variables, 162 clauses, 126 unsat). Figures [1\(e\)](#), [1\(f\)](#) are relative to the Clique instance *johnson8-2-4.clq.cnf* (28 variables, 420 clauses, 420 unsat). A comparison in terms of Kullback-Leibler divergence is presented in table [11](#).

Table 1. Comparison with exact enumerator. Kullback-Leibler divergence between the true density of states and the estimated one.

Instance	variables	clauses	KL-divergence $D_{KL}(n g)$
ram_k3_n7.ra0.cnf	21	70	0.00003956
ram_k3_n8.ra0.cnf	28	126	0.0000119634
johnson8-2-4.clq.cnf	28	420	0.0000458743
t3pm3-5555.spn.cnf	27	162	0.0000130045
Synth. formula (3)	50	100	0.0000118838
Synth. formula (6)	200	750	0.000000125958

benchmark. The true density is computed by exact enumeration. We chose instances that are encodings of three different class of problems (Ramsey Theorem, Spin Glass, Max Clique) and we plotted log-densities because of the large range of values involved.

Although by the effect of the logarithmic scale the two densities in the plots are overlapping to the eye and therefore are not distinguishable, the corresponding relative error plots show that there is small error, that is never greater than 5%. The impressive degree of accuracy obtained is confirmed by the Kullback-Leibler divergences presented in table 1.

We also notice that even though the shape of the DOS is a distinctive characteristic of the original problem class, in all cases the distribution concentrates almost all the probability weight on a small energy range.

5.2 Synthetic Formulas: Exact Analytic Counts

The simplest analytical results can be obtained for a k -SAT formula with m clauses such that each variable appears in exactly one clause (so there are $n = km$ variables). In this case the density of states is

$$n(E) = \binom{m}{E} p^E (1-p)^{m-E} 2^{km} = \binom{m}{E} \left(\frac{1}{2^k}\right)^E \left(1 - \frac{1}{2^k}\right)^{m-E} 2^n, \quad (2)$$

where $p = 1/2^k$ is the probability that a clause is unsatisfied by an assignment chosen uniformly at random.

A more interesting class of instances with a closed form solution can be constructed in the following way:

$$x_1 \wedge x_2 \wedge (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge x_3 \wedge x_4 \wedge (x_3 \vee x_4) \wedge (x_3 \vee \overline{x_4}) \wedge \dots \wedge x_{\ell-1} \wedge x_\ell \wedge (x_{\ell-1} \vee x_\ell) \wedge (x_{\ell-1} \vee \overline{x_\ell}) \quad (3)$$

Each subformula of the form $x_1 \wedge x_2 \wedge (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2})$ has a density of satisfied clauses that is uniform in the interval $[1, 4]$. Using the fact that the probability that the sum of n s -sided dices is k can be written as

$$F[s, n, k] = \frac{1}{s^n} \sum_{i=0}^{\lfloor \frac{k-n}{s} \rfloor} (-1)^i \binom{n}{i} \binom{k-si-1}{n-1}, \quad k > n.$$

We therefore have that the number of configurations satisfying k clauses of a formula constructed as in equation (3) is $F[4, \frac{\ell}{2}, k]2^\ell$ and from that the density of states is

$$n(E) = F[4, \frac{\ell}{2}, 2\ell - E]2^\ell \quad (4)$$

More generally, consider a (small) formula ϕ for which we know the density of states $n_\phi(E)$. We can construct a larger formula F by taking the conjunction of ℓ copies of ϕ , each one involving a different set of variables x_1, \dots, x_ℓ :

$$F(x_1, \dots, x_\ell) = \phi(x_1) \wedge \phi(x_2) \wedge \dots \wedge \phi(x_\ell).$$

Given the independence assumption implied by the fact that by construction the subformulas do not share variables, the DOS $n_F(\cdot)$ of the larger formula F can be obtained in closed form using a multinomial distribution. Moreover, by noticing that the subformulas in F do not share variables, it is easy to see that $n_F(E)$ can be computed as a multiple convolution of $n_\phi(\cdot)$:

$$n_F(E) = (n_\phi * \dots * n_\phi)(E), \quad (5)$$

where $*$ is the convolution operator. This result is analogous to the fact that the probability density function (PDF) of the sum of independent random variables is equal to the convolution of the PDFs of the addends (concentrating the measure on the mean).

In particular, let $P_n(x)$ be the standard CNF encoding of a Pigeon Hole problem with n holes and $n + 1$ pigeons, with $n + 1$ clauses which say that a pigeon has to be placed in some hole and a set of clauses ensuring that only one single pigeon is placed into each hole. This encoding leads to $n(n + 1)$ variables and to $(n + 1) + n(n + 1)/2$ clauses. Now we consider the following CNF formula:

$$P_n^\ell(x_1, \dots, x_\ell) = P_n(x_1) \wedge P_n(x_2) \wedge \dots \wedge P_n(x_\ell) \quad (6)$$

where $x_i \cap x_j = \emptyset$ whenever $i \neq j$. Using (5), the DOS of formula (6) can be obtained as the convolution of the DOS of a single $P_n(x)$ with itself $\ell - 1$ times.

We test the effectiveness of **MCMC-FlatSat** on large synthetic instances, for which exact enumeration would not be possible, by comparing the estimated DOS with the analytical results we just derived. In figure 2(c) and 2(d) we compare the results of **MCMC-FlatSat** on a formula constructed as in equation (3) with the theoretical density of states given by (4). In the experiment presented in figure 2(a) and 2(b) we evaluate the DOS of a single $P_4(x)$ by explicit enumeration, and then we compute the exact DOS of $P_4^{10}(x_1, \dots, x_{10})$ by carrying over the convolutions. This is compared with the approximate result given by **MCMC-FlatSat** when used directly on the large formula (6). Even in this case, the log-densities in the plots are overlapping and therefore are not distinguishable to the eye, and the corresponding relative error is never greater than 3%, as confirmed by the small Kullback-Leibler divergences reported in table 1.

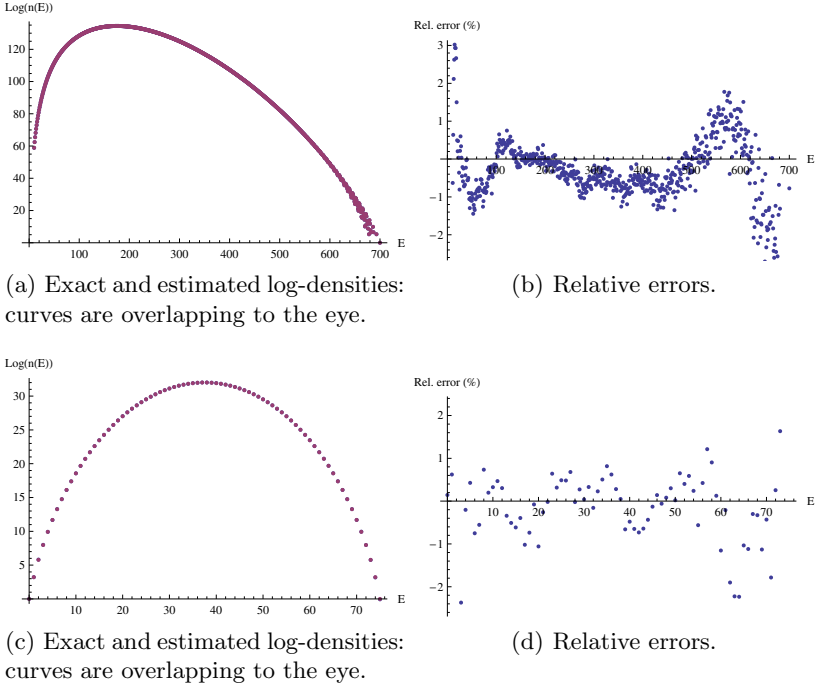
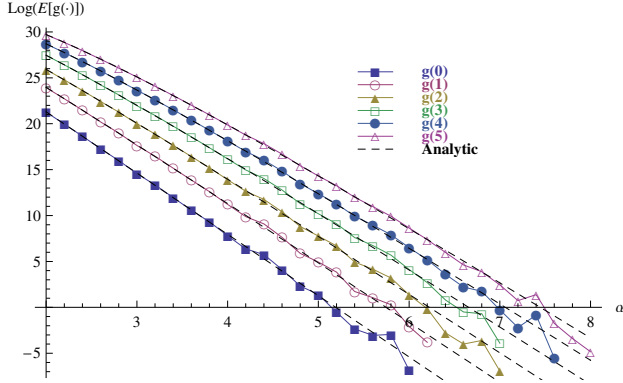


Fig. 2. Comparison of the estimated DOS and the exact analytical results obtained in section 5.2. In figure 2(a) and 2(b) we used a formula constructed as in equation (6) with $n = 4$, $\ell = 10$, resulting in 200 variables, 750 clauses. In figure 2(c) and 2(d) we used a formula constructed as in equation (3) with 50 variables and 100 clauses.

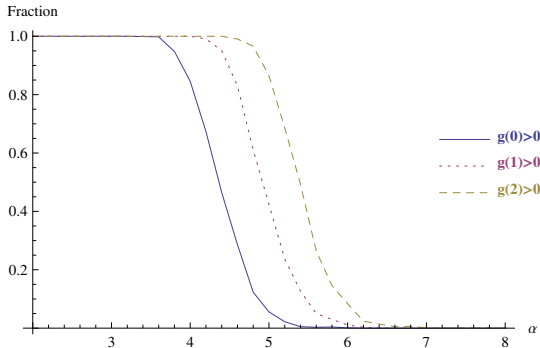
5.3 Random Formulas

In this section we present a detailed study of the behavior of the DOS for random 3-SAT formulas as a function of the ratio clauses to variables α . In particular, we compute the average DOS over 1000 random instances for each value of α in the range considered. By studying the behavior of $g(0)$ (the number of estimated models) in figure 3(a) and 3(b), we recover the well known phase transition between the SAT and UNSAT phase, occurring at the critical ratio $\alpha_c = 4.27$. Notice that we have $\mathbb{E}[g(0)] > 0$ for $\alpha > \alpha_c$ because even if it is true that in that region most of the formulas are not satisfiable, the ones that are contributing to the average with large numbers of solutions (see [11]).

We discovered a similar phase transition behavior for $g(i)$, $i > 0$ as reported in figures 3(a) and 3(b). To the best of our knowledge, this is the first time these phase transition phenomena have been discovered experimentally. Notice however that the average DOS ($\mathbb{E}[g(i)]$) for random k -SAT formulas can be obtained using equation (2). This is because given a truth assignment σ , the probability of having a clause that is violated by σ is $1/2^k$ when the k -SAT formula is chosen uniformly at random. The comparison with the analytic result



(a) Average DOS.



(b) Phase transitions.

Fig. 3. Average DOS and fraction of instances that have $g(i) > 0$ for random 3-SAT formulas as functions of the ratio clauses to variables α . The number of variables is $n = 50$ (see pdf for color version of figures).

(2) in figure 3(a) confirms the good accuracy of the DOS estimation algorithm. Moreover, by using a Markov bound $P[g(i) > 0] \leq \mathbb{E}[g(i)]$ we can get upper bounds on the phase transition thresholds we see in figure 3(b). For instance, we obtain that $P[g(i) > 0] \leq 0.001$ for α greater than 6.22, 6.80, 7.30 for $i = 0, 1, 2$ respectively. Interestingly, using the same Markov bound one can also show that $P[g(i) > 0] \rightarrow 0$ for $\alpha > \log_{8/7} 2 = 5.19\dots$ for $n \rightarrow \infty$ and $i \in o(n/\log(n))$.

With the density of states we can use canonical average formulas to calculate exactly macroscopic properties such as the log-partition function $Z(T)$ at temperature T , defined as $Z(T) = \log \left(\sum_E g(E) e^{-\frac{1}{T} E} \right)$. This property is of considerable theoretical and practical interest because its zero temperature limit $\lim_{T \rightarrow 0} Z(T)$ counts the number of models. Several analytical and algorithmic attempts ([12], [13]) have been made to estimate its value in the low temperature range. Our findings reported in figure 4(a) suggest that small but non-zero

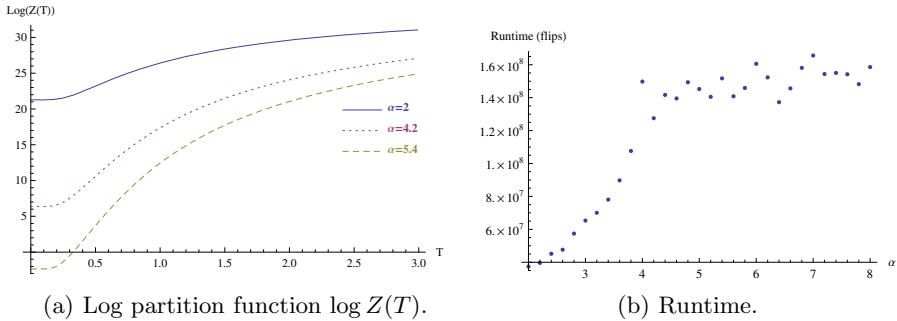


Fig. 4. Log partition function and runtime for random 3-SAT formulas as functions of the ratio clauses to variables and temperature. Notice that the value of $Z(0)$ corresponds to the model count given by $g(0)$ in figure [3\(a\)](#).

temperature approximations of $Z(T)$ can indeed provide accurate information on the number of models for random 3-SAT formulas.

Of practical interest is also the study of the running time of the algorithm presented in figure [4\(b\)](#). We find an increased complexity as we approach the critical threshold α_c that is typical of local search methods. However, given the peculiar nature of this local search method, we can study its behavior even for $\alpha > \alpha_c$. In that range, the runtime increases with a smaller slope, that we believe is caused by the additional effort required to estimate an histogram with an increasing number of energy levels.

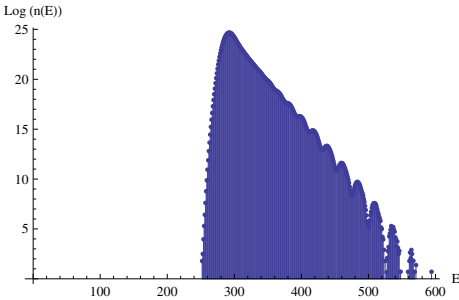
5.4 Large Structured Instances

In this section we present the results obtained on large structured formulas for which the exact DOS is unknown and direct enumeration is not feasible. Given that we are not aware of any complete solver that is able to compute the exact DOS, we need to resort to partial consistency checks to assess the accuracy of **MCMC-FlatSat**. In particular, when it is possible, one can compare $g(0)$ with the exact model count given by a complete solver such as Cachet ([\[14\]](#)). A further consistency check can be obtained by looking at the moments of the DOS. Intuitively, the moments represent a quantitative measure of the shape of a set of points and therefore they can be used to check that the probability mass is concentrated in the right regions. The k -th order moment is defined as

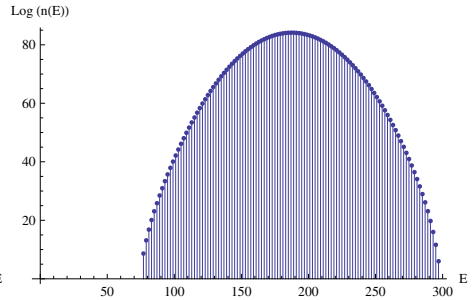
$$M(k) = \sum_E E^k \frac{g(E)}{Z}$$

where $Z = 2^n$ is again used to normalize to a probability distribution. For example, $M(1)$ is the average number of violated clauses by a random assignment. This value is compared with the *sample k -th moment*

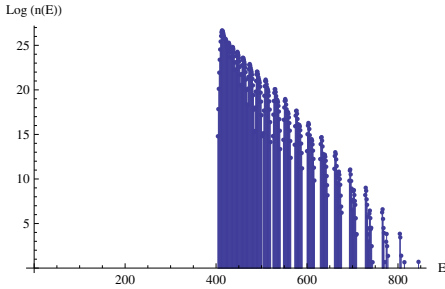
$$M_s(k) = \frac{1}{\ell} \sum_{i=1}^{\ell} E(X_i)^k$$



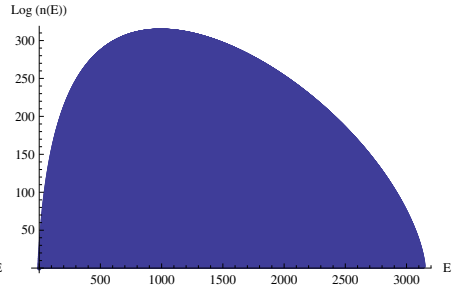
(a) Log-Density for a Clique problem *brock400_2.clq.cnf* from MaxSAT-2009.



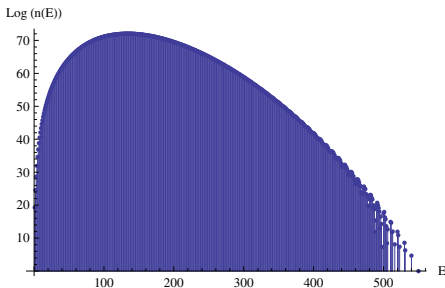
(b) Log-Density for a Spin Glass problem *spinglass5_10.pm3.cnf* from MaxSAT-2009. Notice there are no configurations with an even number of unsatisfied clauses.



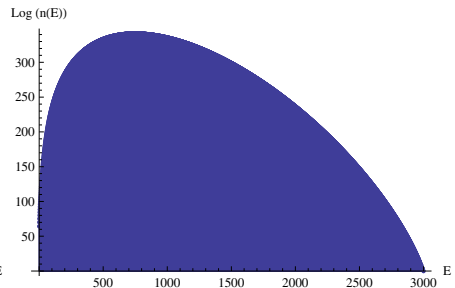
(c) Log-Density for a Clique problem *MANN_a27.clq.cnf* from MaxSAT-2009. No solver presented at MAXSAT09 could solve this instance (within 30 minutes).



(d) Log-Density for the Logistic problem *bw_large.a.cnf* from SATLib.



(e) Log-Density for the Pigeon Hole problem instance *hole10.cnf* from SATLib.



(f) Log-Density for the Morphed Graph Colouring problem *sw100-1.cnf* from SATLib.

Fig. 5. DOS for several large formulas from MaxSAT-2009 and SATLib ([15])

Table 2. Comparison of the moments. Sample moments estimated with $\ell = 10^6$ uniformly sampled truth assignments. Exact model counting is done with Cachet.

Instance	var	clauses	$g(0)$	# models	$M_s(1)$	$M(1)$	$M_s(2)$	$M(2)$
brock400_2.clq.cnf	40	1188	0	0	297.014	297.024	88365.9	88372.3
spinglass5_10.pm3.cnf	125	750	0	0	187.498	187.492	35249.2	35247
MANN_a27.clq.cnf	42	1690	0	0	422.49	422.483	178709	178703
bw_large.a.cnf	459	4675	1	1	995.298	995.322	996349	996634
hole10.cnf	110	561	0	0	137.482	137.562	19621.7	19643.8
sw100-1.cnf	500	3100	8.04×10^{27}		753.072	753.06	571718	571863

where X_1, X_2, \dots, X_ℓ are samples drawn uniformly from all possible assignments. Given that the space of all possible assignments is exponentially large, the samples X_1, X_2, \dots, X_ℓ will be representative only of high probability regions of that space. While this is precisely the reason why the method of uniform sampling cannot be used to estimate an entire DOS, it can still be used to check that the probability mass is concentrated in the right regions.

In figure 5, we present the estimated DOS for several instances from the MaxSAT-2009 benchmark and SATLib ([15]). These kind of results are, to the best of our knowledge, novel. Even though we cannot provide a formal guarantee that our results are accurate, the experimental validation in the previous sections suggests that they should be accurate. Moreover we have a perfect correspondence both with Model Counters and in terms of *sample k -th moments*, as confirmed by the results presented in table 2. In all these instances, we see that the shape of the DOS appears to be a characteristic property of the class of problems that was translated into SAT, and that the probability weight is again concentrated in a small energy range. We believe this type of information can be used to improve local search strategies targeted to a particular class of encodings.

5.5 Model Counting

Even if computing the DOS is a more general and more difficult problem than model counting, comparing MCMC-FlatSat with model counters still provides some useful insights. In particular, we can show that when the number of clauses in the formula is not too big, that is the overhead derived from computing the entire DOS is not overwhelming, MCMC-FlatSat competes against state of the art model counters, both in terms of accuracy and running times. We compared the performance of MCMC-FlatSat with two approximate model counters: Sample-Count ([2]) and SampleMiniSATEXact ([1]). The instances used are taken from the benchmark used in [12]. The results in table 3 show that MCMC-FlatSat generally achieves a greater accuracy, even though it does not provide any guarantee on the results (as [12] do). When the ratio of clauses to variables is not too high, it has comparable if not favorable running times. However, when the number of clauses is too large, the overhead caused by the computation of the entire DOS becomes too large and the comparison in terms of running times becomes unfair, even though it still wins in terms of accuracy.

A more detailed comparison is beyond the scope of this paper, but we believe that a fairly straightforward implementation that forces the random walk to

Table 3. Comparison with model counters on formulas from the benchmark in [2] with a small number of clauses. Timings for SampleCount and SampleMiniSATExact are taken from the respective papers [11,2]. MCMC-FlatSat timings are obtained on a comparable 3Ghz machine.

Instance	n	m	Exact #	SampleCount		SampleMiniSAT		MCMC-FlatSat	
				Models	Time	Models	Time	Models	Time
2bitmax_6	252	766	2.10×10^{29}	$\geq 2.40 \times 10^{28}$	29	2.08×10^{29}	345	1.96×10^{29}	1863
wff-3-3.5	150	525	1.40×10^{14}	$\geq 1.60 \times 10^{13}$	240	1.60×10^{13}	145	1.34×10^{14}	393
wff-3.1.5	100	150	1.80×10^{21}	$\geq 1.00 \times 10^{20}$	240	1.58×10^{21}	128	1.83×10^{21}	21
wff-4-5.0	100	500		$\geq 8.00 \times 10^{15}$	120	1.09×10^{17}	191	8.64×10^{16}	189
ls8-norm	301	1603	5.40×10^{11}	$\geq 3.10 \times 10^{10}$	1140	2.22×10^{11}	168	5.93×10^{11}	2693

stay inside low energy regions, without wasting time exploring the high energy space, could have dramatic impact on model counting. The reason is that the random walk used by estimating DOS is explicitly designed to count, while other sampling-based strategies are built on top of systems that might be too biased towards greedy heuristics when they perform random walks in the configuration space. Moreover, the information collected on how many configurations are not models (that is $g(i), i > 0$) can be effectively used to infer about $g(0)$, given the normalization constraint $\sum g(i) = 2^n$.

6 Conclusions and Future Work

We described MCMC-FlatSat, a Markov Chain Monte Carlo technique based on the flat histogram method to estimate the density of states of Boolean formulas. We demonstrated the effectiveness of MCMC-FlatSat, both in terms of convergence and accuracy, on a broad range of structured and synthetic instances. Using our method, we also provided new insights about the phase transition phenomena of random 3-SAT formulas. We believe that the results presented in this paper are very promising and that the very detailed characterization of the configuration space provided by MCMC-FlatSat will open the way for a new set of heuristics for local search methods, and will provide further insights about random k -SAT formulas as well. Moreover, considering the generality of the flat histogram idea, we expect that this new approach will find many other applications both to counting and inference problems.

Acknowledgments

This research is funded by NSF Expeditions in Computing grant 0832782.

References

1. Gogate, V., Dechter, R.: Approximate counting by sampling the backtrack-free search space. In: Proc. of AAAI-2007, pp. 198–203 (2007)
2. Gomes, C., Hoffmann, J., Sabharwal, A., Selman, B.: From sampling to model counting. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007 (2007)

3. Gomes, C., Sabharwal, A., Selman, B.: Model counting: a new strategy for obtaining good bounds. In: Proceedings of AAAI 2006, pp. 54–61. AAAI Press, Menlo Park (2006)
4. Littman, M., Majercik, S., Pitassi, T.: Stochastic boolean satisfiability. *Journal of Automated Reasoning* 27(3), 251–296 (2001)
5. Sang, T., Beame, P., Kautz, H.: Solving Bayesian networks by weighted model counting. In: Proc. of AAAI 2005, pp. 475–481 (2005)
6. Hansen, P., Jaumard, B.: Algorithms for the maximum satisfiability problem. *Computing* 44(4), 279–303 (1990)
7. Wang, F., Landau, D.: Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters* 86(10), 2050–2053 (2001)
8. Belaidouni, M., Hao, J.K.: Sat, local search dynamics and density of states. In: Selected Papers from the 5th European Conference on Artificial Evolution, pp. 192–204. Springer, Heidelberg (2002)
9. Rosé, H., Ebeling, W., Asselmeyer, T.: The density of states - a measure of the difficulty of optimisation problems. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 208–217. Springer, Heidelberg (1996)
10. Wei, W., Erenrich, J., Selman, B.: Towards efficient sampling: Exploiting random walk strategies. In: Proceedings of the 19th National Conference on Artificial Intelligence, pp. 670–676. AAAI Press, Menlo Park (2004)
11. Kamath, A., Motwani, R., Palem, K., Spirakis, P.: Tail bounds for occupancy and the satisfiability threshold conjecture. In: Proc. of the 35th Annual Symposium on the Foundations of Computer Science, pp. 592–603 (1994)
12. Monasson, R., Zecchina, R.: Entropy of the K-satisfiability problem. *Physical review letters* 76(21), 3881–3885 (1996)
13. Montanari, A., Shah, D.: Counting good truth assignments of random k-SAT formulae. In: Proc. of the 18th ACM Symposium on Discrete Algorithms (2007)
14. Sang, T., Bacchus, F., Beame, P., Kautz, H., Pitassi, T.: Combining component caching and clause learning for effective model counting. In: Proc. of SAT 2004 (2004)
15. Hoos, H., Stützle, T.: SATLIB: An Online Resource for Research on SAT. In: Sat 2000: Highlights of Satisfiability Research in the Year, p. 283 (2000)

Towards Parallel Non Serial Dynamic Programming for Solving Hard Weighted CSP

David Allouche, Simon de Givry, and Thomas Schiex

Unité de Biométrie et Intelligence Artificielle, UR 875, INRA,
F-31320 Castanet Tolosan, France
{david.allouche,simon.degivry,thomas.schiex}@toulouse.inra.fr

Abstract. We introduce a parallelized version of tree-decomposition based dynamic programming for solving difficult weighted CSP instances on many cores. A tree decomposition organizes cost functions in a tree of collection of functions called clusters. By processing the tree from the leaves up to the root, we solve each cluster concurrently, for each assignment of its separator, using a state-of-the-art exact sequential algorithm. The grain of parallelism obtained in this way is directly related to the tree decomposition used. We use a dedicated strategy for building suitable decompositions.

We present preliminary results of our prototype running on a cluster with hundreds of cores on different decomposable real problems. This implementation allowed us to solve the last open CELAR radio link frequency assignment instance to optimality.

1 Introduction

Graphical model processing is a central problem in AI. The optimization of the combined cost of local cost functions, central in the valued CSP framework [12], captures problems such as weighted Max-SAT, Weighted CSP or Maximum Probability Explanation in probabilistic networks. It has applications in *resource allocation* [2], *combinatorial auctions*, *optimal planning*, *bioinformatics*. Valued constraints can be used to code classical crisp constraints and cost functions.

Because these problems are NP-hard, however, there are always relevant problems which cannot be solved in reasonable time. With the current trend of increasing number of cores per machine and increasing number of machines in clusters or grids, it is only natural to try to exploit problem decomposability by distributing the workload on a large number of computing resources.

In this paper, we use tree decompositions as a source of workload distribution. Tree decomposition has repeatedly been used to solve reasoning problems in graphical models, from constraint satisfaction [4] to bayesian networks [7]. In constraint satisfaction, different algorithms such as Adaptive Consistency, Bucket Elimination or Cluster Tree Elimination rely on a tree decomposition to decompose the original problem and improve the worst-case time complexity.

On a single core, our algorithm can be described as a block by block elimination process in non serial dynamic programming as proposed in [1]. In a CSP

context, it can also be described as a simplified version of Cluster Tree Elimination [5] that performs only one pass over the problem.

We show how the tree decomposition can be exploited to distribute the workload on many cores and how the granularity of the distribution can be controlled by modifying the tree decomposition itself. We define specific operations on tree decompositions allowing to reach a suitable tree decomposition, when it exists. We then perform an empirical evaluation of the algorithm on different hard instances coming from bioinformatics (haplotyping problems) and resource allocation (CELAR radio frequency assignment problems) and analyze the influence of the number of cores on the time needed to solve the problem.

2 Background

A weighted CSP (WCSP) is a pair (X, W) where $X = \{1, \dots, n\}$ is a set of n variables and W a set of cost functions. Each variable $i \in X$ has a finite domain D_i of values than can be assigned to it. The maximum domain size is d . For a set of variables $S \subseteq X$, D_S denotes the Cartesian product of the domain of the variables in S . For a given tuple of values t , $t[S]$ denotes the projection of t over S . A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D_S \mapsto [0, k]$ where k is a maximum integer cost used for forbidden assignments (expressing hard constraints). It can be described by a table or by an analytic function.

Cost functions can be manipulated by two operations. For two cost functions w_S and $w_{S'}$, the *combination* $w_S \oplus w_{S'}$ is a cost function $w_{S \cup S'}$ defined as $w_{S \cup S'}(t) = w_S(t[S]) + w_{S'}(t[S'])$. The *marginal* $w_S \downarrow_{S'}$ of w_S on $S' \subset S$ is a cost function over S' defined as $w_S \downarrow_{S'}(t') = \min_{t \in D_S, t[S'] = t'} w_S(t)$ for all $t' \in D_{S'}$.

The weighted Constraint Satisfaction Problem is to find a complete assignment t minimizing the combined cost function $\sum_{w_S \in W} w_S(t[S])$. This optimal cost can be defined using combination and marginalization as $(\bigoplus_{w_S \in W} w_S) \downarrow_{\emptyset}$. This optimization problem has an associated NP-complete decision problem.

The hypergraph of a WCSP is an hypergraph $H = (X, E)$ with one vertex for each variable and one hyperedge S for every cost function $w_S \in W$. The primal graph of H is an undirected graph $G = (X, F)$ s.t. there is an edge $(i, j) \in F$ for any two vertices $i, j \in X$ that appear in the same hyperedge S in H .

A tree decomposition for a WCSP (X, W) is a triple (T, χ, ψ) where $T = (V, E)$ is a tree and χ, ψ are labelling functions that associate with each vertex $v \in V$ a set of variables $\chi(v) \subseteq X$ and a set of cost functions $\psi(v) \subseteq W$ s.t.:

1. for each $w_S \in W$, there is exactly one vertex $v \in V$ s.t. $w_S \in \psi(v)$
2. if $w_S \in \psi(v)$ then $S \subseteq \chi(v)$
3. $\forall i \in X$, the set $\{v \in V \mid i \in \chi(v)\}$ induces a connected subtree of T .

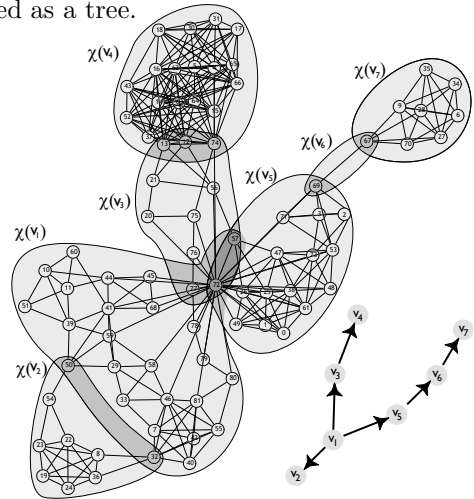
The treewidth of a tree decomposition, denoted by w , is the size of the largest set $\chi(v)$ minus one. For a given $v \in V$, the WCSP $(\chi(v), \psi(v))$ is a subproblem of the master WCSP (X, W) . Two subproblems defined by two vertices in the tree may share variables, but not cost functions (property [II]). A tree decomposition can be rooted by choosing a root $v \in V$. For a vertex $v \in V$, the separator of v

is $s(v) = \chi(v) \cap \chi(\text{father}(v))$ (where $\text{father}(v)$ is the father of v in T or \emptyset for the root). The variables of $\chi(v) \setminus s(v)$ are said to be the *proper* variables of v .

The rationale behind tree decompositions is that acyclic problems can be solved with a good theoretical complexity as it decomposes a master problem in subproblems (called clusters) organized as a tree.

This is illustrated on the right, where the graph of a frequency assignment problem is covered by clusters (the sets $\chi(v)$) defining a tree decomposition. Locality of information in space or time often yield such nice decompositions in real problems.

Different algorithms have been proposed to exploit such tree decompositions. We focus in this paper on *elimination algorithms* such as variable or bucket elimination (BE) [3,11]. In BE, a variable is chosen, all the cost functions that involve this variable are removed from the problem,



and the marginal of the combination of these functions on the rest of the problem is added to the original problem. The problem obtained has one less variable and the same optimal cost as the original problem [3]. By repeatedly eliminating all variables, we ultimately get the optimal cost. One can show [5] that an implicit rooted tree decomposition lies behind BE and the complexity of the whole elimination process depends on this tree decomposition.

Instead of eliminating one variable after the other, one may eliminate several variables at once. This is called block-by-block elimination [1]. Typically, all the proper variables of a leaf cluster $v \in T$ are eliminated in one step. This is done repeatedly until the root cluster is eliminated, leaving the optimal cost to read as a constant. This approach has been sophisticated in Cluster Tree Elimination (CTE [5]) which may compute the marginals for each variable in two passes. Such algorithms have been parallelized in the context of inference, with no pruning [13].

3 Parallelization of Block by Block Elimination

In this paper, since we are just interested in computing the global optimum, we use the one pass block-by-block elimination algorithm described in [1] for optimization and adapt it to WCSP. Given an initial WCSP (X, W) together with a tree decomposition (T, χ, ψ) of the WCSP, the elementary action of the algorithm is to *process* a leaf cluster $v \in T$, by eliminating all its proper variables:

1. we compute the marginal cost function F on the separator $s(v)$:

$$F = \left(\bigoplus_{w_S \in \psi(v)} w_S \right) \downarrow_{s(v)}$$

2. we remove v from T , $\psi(v)$ from W and all proper variables of v from X
3. we add F to both $\psi(\text{father}(v))$ and W

As in BE, the problem obtained has less variables and the same optimum [1] and the updated triple (T, χ, ψ) is a tree decomposition of this problem. The process can be repeated on any leaf of the new tree T until an empty tree is obtained. The last marginal function F computed is a constant (a marginal on an empty set of variables) and its value is the optimum of the problem.

Theorem 1. *The time complexity for processing one cluster $v \in T$ is $O((|\psi(v)| + \#sons).d^{|\chi(v)|})$ and the space complexity is $O(d^{|\psi(v)|})$ where d is the domain size and $\#sons$ the number of sons of v in the original T .*

Proof. There are at most $d^{|\chi(v)|}$ costs to compute and $((|\psi(v)| + \#sons)$ cost functions in v . For space, there are $d^{|\psi(v)|}$ costs to store for the projection.

The overall sequential complexity is obtained by summing on all clusters leading to a time complexity of $O((e + |T|).d^w)$ and a space complexity of $O(|T|.d^r)$ where w is the treewidth and $r = \max_v(|s(v)|)$ is the maximum separator size. To parallelize this algorithm, two sources of non determinism can be exploited.

- The first lies in the computation of the marginal cost function F . It can be computed independently for each possible assignment t of the separator $s(v)$ by assigning the separator variables with t in the WCSP $(\chi(v), \psi(v))$ and solving the WCSP obtained using a state-of-the-art DFBB algorithm.
- the second source of non determinism lies in the fact that the tree T may have several leaves. Whenever all the sons of a vertex $v \in T$ have been processed, v becomes a leaf and can be processed.

To get an idea of the speedups that can be reached in ideal (inexistent) conditions, we consider a PRAM machine with an unbounded number of cores, a constant time memory access, with a constant sequential setup time “ s “ for starting a new process.

Theorem 2. *With an unbounded number of cores, the time complexity for processing one cluster v is $O((|\psi(v)| + \#sons).d^{|\chi(v) - s(v)|}) + s.d^{|\psi(v)|})$ and the space complexity is $O(d^{|\psi(v)|})$.*

Proof. The complexity of solving $(\chi(v), \psi(v))$ after its separator assignment is $O((|\psi(v)| + \#sons).d^{|\chi(v) - s(v)|})$ since only $|\chi(v) - s(v)|$ variables remain unsigned. In parallel, the complete projection can be done in the same bound. The extra setup time for the $d^{|\psi(v)|}$ jobs is $O(s.d^{|\psi(v)|})$.

Since a cluster v cannot be processed until all its sons have been processed, the overall complexity is controlled by the longest path from the root to any leaf where the length of a path is the sum of the processing times of its clusters.

Compared to the sequential complexity, one see the impact of the sources of parallelism exploited. The separator assignment based parallization shifts complexity from an exponential in w (max. cluster size) to one in the maximum number of proper variables but we get an additive (instead of multiplicative) exponential term in r (max. separator size). The use of the T itself as a source of parallelism replaces a factor related to tree size by a factor related to its height.

4 Producing a Suitable Tree Decomposition

There has been a lot of work on tree decompositions. Usually, the problem considered is to produce a decomposition with a minimum treewidth, an NP-hard problem. We used Min-Fill and MCS heuristics, very usual heuristics aimed at the production of tree-decompositions with small treewidth [10].

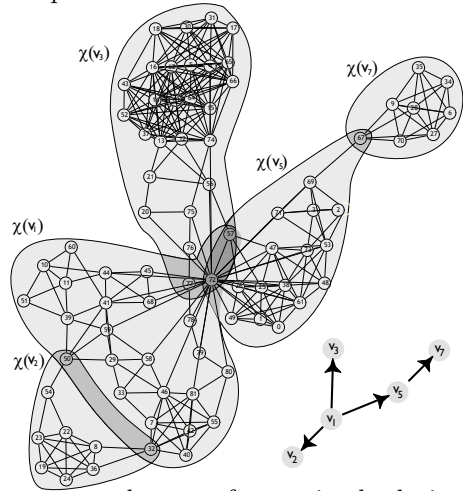
Because of space complexity, separator size, instead of treewidth, is the main restricting factor for elimination algorithms. In our case, this is reinforced by the influence of setup times on time complexity. More precisely, for a separator $s(v)$, the limiting factor is the associated separator space $SS(v)$, defined as the size of the Cartesian product of the domains. These can be summed up over all $v \in T$ to compute the overall separator space (and number of jobs) associated with the decomposition. We use decompositions with a bounded separator space.

A traditional approach to get rid of a large separator is the so-called “super-bucket” approach [5] which simply consists in merging the two separated clusters v and $father(v)$ into one if the separator $s(v)$ contains more than r_{\max} variables.

From the theoretical parallel complexity, one should *a priori* favor decompositions with a small maximum number of proper variables. However, time complexity is controlled by a tight ($s.d^r$) term for setup times and a much looser upper bound in $O(d^p)$ for solving assigned problems.

Therefore, the number of proper variables should not be too small or the overall time will be dominated by setup times. We therefore extend the “super-bucket” strategy, merging a cluster v with $father(v)$ whenever the number of proper variables of v is smaller than a bound x_{\min} .

In the previously shown tree decomposition, we merge v_3 and v_4 because the separator space is too large, and similarly we merge v_5 and v_6 because v_6 has too little proper variables, leading to the updated tree decomposition in the figure on the right.



Note that our elimination algorithm computes the cost of an optimal solution, but not the optimal solution itself. Producing an optimal solution afterwards can be done by solving each cluster once from root to leaves, using the projection of the optimal solution found for the father cluster on the separator of the current cluster as the initial values of the separator variables of the son. This takes less time than calculating the minimum cost itself.

5 Implementation and Results

To evaluate the practical interest of parallel block-by-block elimination, we have experimented it on hard decomposable problems. We preprocess a WCSP by

eliminating all variables with a small degree (2 or 3). An initial tree decomposition is produced by the Min-Fill and MCS heuristics [10]. For all values of r_{\max} and x_{\min} in $[1, 25]$ and $[1, 33]$ respectively, the super-bucket strategy is applied and the corresponding separator space SS and treewidth w are computed. If no decomposition with $SS \leq 2 \cdot 10^6$ exists, the method is said to be unsuitable for this instance. Else, we select a decomposition with minimum treewidth, unless an alternative decomposition exists that has a treewidth close to the minimum and a significantly smaller separator space (used for instances with a * in the table below). The decomposition is rooted in the cluster with maximum $|\chi(v)|$.

This process has been applied to the CELAR radio link frequency assignment instances `scen06`, `scen07`, and `scen08` (still open) and to hard genetics haplotyping instances, which have been tackled by a parallel AND/OR branch and bound approach in [9].

The table on the right gives, for each instance, the number of variables, maximum domain size, cost functions, treewidth, number of clusters and separator space obtained after preprocessing and “super-bucketing”

Name	$ X $	d	$ W $	w	$ T $	SS	r_{\max}	x_{\min}
scen06*	82	44	409	40	7	$3 \cdot 10^5$	3	3
scen07*	162	44	927	56	5	$4 \cdot 10^4$	3	13
scen08	365	44	1905	85	13	$2 \cdot 10^6$	4	15
ped7*	293	4	667	141	28	$2 \cdot 10^5$	14	5
ped19*	484	5	1092	79	40	$1 \cdot 10^6$	15	7
ped31*	513	5	1106	132	300	$4 \cdot 10^5$	12	1

with corresponding r_{\max} and x_{\min} (using MCS for scen* and Min-Fill for ped*).

The resolution of the WCSP $(\chi(v), \psi(v))$ with a specific separator assignment is done using a Depth First Branch and Bound (DFBB) algorithm maintaining Existential Directional Arc Consistency [6] with a variable ordering combining Weighted Degree with Last Conflict [8] and using an upper bound produced by an initial restart strategy using Limited Discrepancy Search with a randomized variable ordering. Cost functions of arity above 3 are preprojected onto binary and ternary scopes. This corresponds to executing `toulbar2`¹ with options `hLL`.

The parallel elimination process has been performed on a cluster with 400 cores (L5420 Xeon, 2.5 GHz with 4GB RAM per core) managed by the Sun Grid Engine. SGE uses a “fair share” strategy which may drastically change the number of cores allocated during execution. This makes a precise analysis of the effect of parallelization impossible. We therefore measured the CPU-time, number of nodes and backtracks of every `toulbar2` execution. These times are used as input to a k -core simulator using real individual CPU- times. This simulation is played for different numbers of cores ($k = 1, 15, 100, 1,000$) using a 10 second job setup time for multi-core executions *only* (this is the `scheduler_interval` used in the SGE configuration and it is consistent with real time: scen08 took less than 2 days to solve). Each job is in charge of solving 100 separator assignments. For two instances, solved with 100 reserved cores, we report the wall-clock time (inside parenthesis). We then give the CPU-time (on the same hardware, 1 core) of BTD-RDS [11], a state-of-the-art sequential branch and bound algorithm exploiting the same tree decomposition. A “-” indicates the problem could not be

¹ <http://mulcyber.toulouse.inra.fr/projects/toulbar2> version 0.9.1.2.

solved after 7 days. Finally, lower/upper bounds on the time needed to *rebuild* a solution on 1 core (sum of the min/max time in each cluster) is given.

Name	1 core	15 cores	10 ² cores	10 ³ cores	BTD-RDS	rebuild (l/u)
scen06	13h 08'	1h 28'	15' 39" (20' 53")	3' 32"	1' 26"	2' 44"/2' 47"
scen07	1d 6h	2h 09'	23' 27" (42' 02")	10' 07"	-	3' 08"/3' 10"
scen08	127d 14h	8d 15h	1d 7h	3h 13'	-	1h 32'/1h 37'
ped7	40' 52"	22' 55"	3' 37"	35"	1' 6"	11"/11"
ped19	30d 3h	2d 2h	7h 32'	58' 18"	1d, 12h	1' 17"/1' 47"
ped31	20' 27"	41' 03"	6' 55"	1' 25"	3h 37'	44"/45"

For the smallest scen06 problem, which is easily solved using BTD-RDS, the approach is counter productive. For harder RLFAP problems, block elimination is capable of exploiting the full power of parallelism, enabling us to solve for the first time scen08 to optimality. The haplotyping problem instances ped7, 19 and 31 have been selected among the hard instances solved in [9] using a parallel AND/OR branch and bound with mini-buckets on similar CPUs. Our decomposition strategy gives again good results. If ped7 is too simple (less than 2' with BTD-RDS) to benefit from parallelization, for ped19 and ped31, we obtain good results. The elimination algorithm can be slower on 15 cores than on 1 core because of the 10" setup times. This is the case for ped31, because the 100 assignment jobs have a duration which is very short compared to setup time. In this case, a large part of the gain comes from the elimination algorithm itself, which is more efficient than BTD-RDS even in a sequential context. These times can be compared, with caution (these are simulated executions), with the best parallel (15 cores) times of [9]: 53'56", 8h59' and 2h09' respectively: elimination, which is restricted to decomposable problems, gives better results on 100 cores. Contrarily to [9], which observed a large variability in the job durations, we observed a relatively limited variability of job durations inside a cluster. This could be explained by the small separator sizes (the different problems solved for a given cluster are not so different) and the upper-bounding procedure that adapts to each assignment. Note that sufficient separator space is needed to avoid starvation which occurs for example in scen07 on 1000 cores: scen07 defines only 400 jobs of 100 assignments. However, on the problems tested, job granularity seems relatively well handled by our super-bucket strategy. It could be improved by tuning the number of assignments per jobs according to the separator space, number of cores, tree decomposition topology and mean job duration.

6 Conclusion and Future Work

This paper presents a parallelization of an elimination algorithm for WCSP. The approach is suitable for difficult problems that can be decomposed into a tree-decomposition with small separator size. This can be tested beforehand. Two sources of parallelization have been used in this process: the dominating one is based on partial problem assignments in separators, the second one comes from the branching of the tree-decomposition itself. The application of our prototype

on different real hard decomposable problems shows that a large number of cores can be effectively exploited to solve hard WCSP. This allowed us to solve to optimality the last CELAR radio link frequency assignment problem open instance.

Obviously, more extensive experiments are needed on other decomposable problems. Beyond this, the most obvious way to improve the method is to tune the granularity of the jobs processed. We have used the super-bucket strategy and a fixed number of assignment solved per job to reach a suitable granularity, with good results on hard problems. Such tuning should ideally be done based on the problem at hand or even better, dynamically, as it is done in Branch and Bound parallelization. Existing B&B parallelization strategies could also be exploited inside each cluster resolution. Ultimately smarter scheduling strategies, taking into account precedences between clusters, could further improve the overall efficiency of the implementation.

Acknowledgments. This work is partly supported by Toulouse Midi-Pyrénées bioinformatic platform.

References

1. Bertelé, U., Brioshi, F.: Nonserial Dynamic Programming. Academic Press, London (1972)
2. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.: Radio link frequency assignment. *Constraints* 4, 79–89 (1999)
3. Dechter, R.: Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113(1-2), 41–85 (1999)
4. Dechter, R., Pearl, J.: Tree clustering for constraint networks. *Artificial Intelligence* 38, 353–366 (1989)
5. Kask, K., Dechter, R., Larrosa, J., Dechter, A.: Unifying Cluster-Tree Decompositions for Reasoning in Graphical models. *Artificial Intelligence* 166(1-2), 165–193 (2005)
6. Larrosa, J., de Givry, S., Heras, F., Zytnicki, M.: Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In: *Proc. of the 19th IJCAI*, Edinburgh, Scotland, pp. 84–89 (August 2005)
7. Lauritzen, S., Spiegelhalter, D.: Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society – Series B* 50, 157–224 (1988)
8. Lecoutre, C., Saïs, L., Tabary, S., Vidal, V.: Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence* 173, 1592–1614 (2009)
9. Otten, L., Dechter, R.: Towards parallel search for optimization in graphical models. In: *Proc. of ISAIM 2010*. Fort Lauderdale (FL), USA (2010)
10. Rose, D.: Tringulated graphs and the elimination process. *Journal of Mathematical Analysis and its Applications* 32 (1970)
11. Sanchez, M., Allouche, D., de Givry, S., Schiex, T.: Russian doll search with tree decomposition. In: *Proc. IJCAI 2009*, San Diego, CA, USA, pp. 603–608 (2009)
12. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In: *Proc. of the 14th IJCAI*, Montréal, Canada, pp. 631–637 (August 1995)
13. Xia, Y., Prasanna, V.: Parallel Exact Inference. In: Bischof, C., Bücker, M., Gibbon, P. (eds.) *Parallel Computing 2007*. NIC, vol. 38, pp. 185–192. John von Neumann Institute for Computing (September 2007)

Making Adaptive an Interval Constraint Propagation Algorithm Exploiting Monotonicity

Ignacio Araya, Gilles Trombettoni, and Bertrand Neveu

UTFSM (Chile), INRIA, Univ. Nice–Sophia, Imagine LIGM Univ. Paris–Est (France)
iaraya@inf.utfsm.cl, {Gilles.Trombettoni,Bertrand.Neveu}@sophia.inria.fr

Abstract. A new *interval* constraint propagation algorithm, called *MOnotonic Hull Consistency (Mohc)*, has recently been proposed. *Mohc* exploits monotonicity of functions to better filter variable domains. Embedded in an interval-based solver, *Mohc* shows very high performance for solving systems of numerical constraints (equations or inequalities) over the reals. However, the main drawback is that its *revise* procedure depends on two user-defined parameters.

This paper reports a rigorous empirical study resulting in a variant of *Mohc* that avoids a manual tuning of the parameters. In particular, we propose a policy to adjust in an auto-adaptive way, during the search, the parameter sensitive to the monotonicity of the revised function.

1 Introduction

Interval-based solvers can solve systems of numerical constraints, i.e., nonlinear equations or inequalities over the reals. Their reliability and increasing performance make them applicable to various domains such as robotics design [10], dynamic systems in robust control or robot localization [8], robust global optimization [7,12] and bounded-error parameter estimation [6].

To find all the solutions of a numerical CSP, an interval-based solving strategy starts from an initial search space called a *box* (an interval for every variable in the handled system) and builds a search tree by following a *Branch & Contract* scheme. Filtering or *contraction* algorithms reduce the search space, i.e., improve the bounds of the corresponding variable intervals, with no loss of solution. *Mohc* is a new contraction algorithm based on interval constraint propagation [2]. *Mohc-Revise* exploits monotonicity of functions to improve contraction/filtering. Monotonicity is generally verified for a few functions at the top of the search tree, but can be detected for more functions when smaller boxes are handled. In practice, experiments shown in [2,1] highlight very high performance of *Mohc*, in particular when it is used inside 3BCID [13]. The combination 3BCID(*Mohc*) appears to be state of the art in this area.

The *Mohc-Revise* algorithm handles two user-defined parameters falling in $[0,1]$: a precision ratio ϵ and another parameter called τ_{mohc} that reflects a sensitivity to the degree of monotonicity of the handled function f . This paper mainly presents a rigorous empirical study leading to a policy for tuning τ_{mohc} in an adaptive way for every function f . An adjustment procedure is run in a

preprocessing phase called at every node of the search tree. Its cost is negligible when Mohc is called in combination with 3BCID.

2 Intervals and Numerical CSPs

Intervals allow reliable computations on computers by managing floating-point bounds and outward rounding.

Definition 1 (Basic definitions, notations)

\mathbb{IR} denotes the set of **intervals** $[v] = [a, b] \subset \mathbb{R}$, where a , also denoted \underline{v} , and b , also denoted \overline{v} , are floating-point numbers.

$\text{Diam}([v]) := \overline{v} - \underline{v}$ denotes the **size**, or *diameter*, of $[v]$.

A **box** $[V] = ([v_1], \dots, [v_n])$ represents the Cartesian product $[v_1] \times \dots \times [v_n]$.

Interval arithmetic has been defined to extend to \mathbb{IR} elementary functions over \mathbb{R} [11]. For instance, the interval sum is defined by $[v_1] + [v_2] = [v_1 + v_2, \overline{v_1} + \overline{v_2}]$. When a function f is a composition of elementary functions, an **extension** of f to intervals \mathbb{IR} must be defined to ensure a *conservative* image computation. The **natural extension** $[f]_N$ of a real function f replaces the real arithmetic by the interval arithmetic. The *monotonicity-based extension* is particularly useful in this paper. A function f is *monotonic* w.r.t. a variable v in a given box $[V]$ if the evaluation of the partial derivative of f w.r.t. v is positive (resp. negative) or null in every point of $[V]$. For the sake of conciseness, we sometimes write that *the variable v is monotonic*.

Definition 2 (f_{min} , f_{max} , monotonicity-based extension)

Let f be a function defined on variables V of domains $[V]$. Let $X \subseteq V$ be a subset of monotonic variables. Consider the values x_i^+ and x_i^- such that: if $x_i \in X$ is an increasing (resp. decreasing) variable, then $x_i^- = \underline{x}_i$ and $x_i^+ = \overline{x}_i$ (resp. $x_i^- = \overline{x}_i$ and $x_i^+ = \underline{x}_i$). Consider $W = V \setminus X$ the set of variables not detected monotonic. Then, we define

$$\begin{aligned} f_{min}(W) &= f(x_1^-, \dots, x_n^-, W) \\ f_{max}(W) &= f(x_1^+, \dots, x_n^+, W) \end{aligned}$$

Finally, the *monotonicity-based extension* $[f]_M$ of f in the box $[V]$ produces the interval image $[f]_M([V]) = \left[[f_{min}]_N([W]), [f_{max}]_N([W]) \right]$.

Consider for example $f(x_1, x_2, w) = -x_1^2 + x_1x_2 + x_2w - 3w$.

$[f]_N([6, 8], [2, 4], [7, 15]) = -[6, 8]^2 + [6, 8] \times [2, 4] + [2, 4] \times [7, 15] - 3 \times [7, 15] = [-83, 35]$. $\frac{\partial f}{\partial x_1}(x_1, x_2) = -2x_1 + x_2$. Since $[\frac{\partial f}{\partial x_1}]_N([6, 8], [2, 4]) = [-14, -8] < 0$, we deduce that f is decreasing w.r.t. x_1 . With the same reasoning, x_2 is increasing and w is not detected monotonic. Following Def. 2, the monotonicity-based evaluation yields:

$$[f]_M([V]) = \left[[f]_N(\underline{x}_1, \underline{x}_2, [w]), [f]_N(\underline{x}_1, \overline{x}_2, [w]) \right] = \left[[f]_N(8, 2, [7, 15]), [f]_N(6, 4, [7, 15]) \right] = [-79, 27]$$

The **dependency problem** is the main issue of interval arithmetic. It is due to **multiple occurrences** of a same variable in an expression that are handled as different variables by interval arithmetic and produce an overestimation of the interval image. Since the monotonicity-based extension replaces intervals by bounds, it suppresses the dependency problem for the monotonic variables. It explains why the interval image computed by $[f]_M$ is sharper than or equal to the one produced by $[f]_N$.

This paper deals with nonlinear systems of constraints or numerical CSPs.

Definition 3 (NCSP). A numerical CSP $P = (V, C, [V])$ contains a set of constraints C , a set V of n variables with domains $[V] \in \mathbb{IR}^n$. A solution $S \in [V]$ to P satisfies all the constraints in C .

The interval-based solving strategies follow a *Branch & Contract* process to find all the solutions of an NCSP. We present two existing **interval constraint propagation** algorithms for contracting the current box in the search tree with no loss of solution. HC4 and Box [3,14] perform a constraint propagation loop and differ in their *revise* procedure handling the constraints individually. The procedure HC4-Revise traverses twice the tree representing the mathematical expression of the constraint for narrowing all the involved variable intervals. Since the different occurrences of a same variable are handled as different variables, HC4-Revise is far from bringing an optimal contraction (dependency problem). The BoxNarrow revise procedure of Box is more costly and stronger than HC4-Revise [5]. For every pair (f, x) , where f is a function of the considered NCSP and x is a variable involved in f , BoxNarrow first replaces the other a variables in f by their interval $[y_1], \dots, [y_a]$. Then, it uses a *shaving* principle where slices $[x_i]$ at the bounds of $[x]$ that do not satisfy the constraint are eliminated from $[x]$. BoxNarrow is not optimal in case the variables y_i different from x also have multiple occurrences.

These algorithms are used in our experiments as *sub-contractor* of 3BCID [13], a variant of 3B [9]. 3B uses a shaving refutation principle that splits an interval into slices. A slice at the bounds is discarded if calling a sub-contractor (e.g., HC4) on the resulting subproblem leads to no solution.

3 Overview of the Mohc Algorithm

The *MO*notonic *H*ull-*C*onsistency algorithm (in short **Mohc**) is a recent constraint propagation algorithm that exploits monotonicity of functions to better contract a box [2]. The contraction brought by Mohc-Revise is optimal (with a precision ϵ) when f is monotonic w.r.t. every variable x involved in f in the current box. Mohc has been implemented with the interval-based C++ library Ibex [4]. It follows a standard propagation loop and calls the Mohc-Revise procedure for handling one constraint $f(V) = 0$ individually (see Algorithm 1).

Mohc-Revise starts by calling HC4-Revise. The monotonicity-based contraction procedures (i.e., MinMaxRevise and MonotonicBoxNarrow) are then called only if V contains at least one variable that appears several times (function

Algorithm 1. Mohc-Revise (in-out $[V]$; in $f, V, \rho_{mohc}, \tau_{mohc}, \epsilon$)

```

HC4-Revise ( $f(V) = 0, [V]$ )
if MultipleOccurrences( $V$ ) and  $\rho_{mohc}[f] < \tau_{mohc}$  then
  ( $X, Y, W, f_{max}, f_{min}, [G]$ )  $\leftarrow$  PreProcessing( $f, V, [V]$ )
  MinMaxRevise ( $[V], f_{max}, f_{min}, Y, W$ )
  MonotonicBoxNarrow ( $[V], f_{max}, f_{min}, X, [G], \epsilon$ )
end if

```

MultipleOccurrences). The other condition is detailed below. The procedure PreProcessing computes the gradient of f (stored in the vector $[G]$) and determines the two functions f_{min} and f_{max} , introduced in Definition 2, that exploit the monotonicity of f . The gradient is also used to partition the variables in V into three subsets X , Y and W :

- variables in X are monotonic and occur several times in f ,
- variables in Y occur once in f (they may be monotonic),
- variables $w \in W$ appear several times in f and are *not* detected monotonic, i.e., $0 \in [\frac{\partial f}{\partial w}]_N([V])$. (They might be monotonic – due to the overestimation of the evaluation – but are considered and handled as non monotonic.)

The next two routines are in the heart of Mohc-Revise. Using the monotonicity of f_{min} and f_{max} , MinMaxRevise contracts $[Y]$ and $[W]$ while MonotonicBoxNarrow contracts $[X]$. MinMaxRevise is a monotonic version of HC4-Revise. It applies HC4-Revise on the two constraints: $f_{min}([Y \cup W]) \leq 0$ and $0 \leq f_{max}([Y \cup W])$. MonotonicBoxNarrow performs a loop on every monotonic variable $x \in X$. If f is increasing w.r.t. x , it performs a first binary search with f_{max} to improve \underline{x} , and a second one with f_{min} to improve \bar{x} . A binary search runs in time $O(\log(\frac{1}{\epsilon}))$, where ϵ is a user-defined precision parameter expressed as a percentage of the interval size.

The user-defined parameter $\tau_{mohc} \in [0, 1]$ allows the monotonicity-based procedures to be called more or less often during the search (see Algorithm 1). For every constraint, the procedures exploiting monotonicity of f are called only if $\rho_{mohc}[f] < \tau_{mohc}$. The ratio $\rho_{mohc}[f] = \frac{Diam([f]_M([V]))}{Diam([f]_N([V]))}$ indicates whether the monotonicity-based image of a function is sufficiently sharper than the natural one. $\rho_{mohc}[f]$ is computed in a preprocessing procedure called after every bisection/branching. Since more cases of monotonicity occur on smaller boxes, Mohc-Revise activates in an adaptive way the machinery related to monotonicity. This ratio is relevant for the bottom-up evaluation phases of MinMaxRevise, and also for MonotonicBoxNarrow in which a lot of evaluations are performed.

4 Making Mohc Auto-Adaptive

The procedure Mohc-Revise has two user-defined parameters, whereas Box-Narrow has one parameter and HC4-Revise has no parameter. The goal of this paper is to avoid the user fixing the parameters of Mohc-Revise.

All the experiments have been conducted on the same Intel 6600 2.4 GHz over 17 NCSPs with a finite number of zero-dimensional solutions issued from COPRIN's web page maintained by J.-P. Merlet.¹ Ref. [2] details the criteria used to select these NCSPs with multiple occurrences.² All the solving strategies use a round-robin variable selection. Between two branching points, three procedures are called in sequence. First, a monotonicity-based existence test [1] cuts the current branch if the image computed by a function does not contain zero. Second, the evaluated contractor is called: `3BCID(Mohc)` or `3BCID(Amohc)` where `Amohc` denotes an auto-adaptive variant of `Mohc`. Third, an interval Newton is run. The shaving precision ratio in `3BCID` is 10% ; a constraint is pushed into the propagation queue if the interval of one of its variables is reduced more than 10%.

We have first studied how the ratio $\frac{\text{Time}(3BCID(\text{Mohc}))}{\text{Time}(3BCID(\text{LazyMohc}))}$ evolves when ϵ decreases, i.e., when the required precision in `MonotonicBoxNarrow` increases. `LazyMohc` [1] is a variant of `Mohc` that does not call `MonotonicBoxNarrow`. The measured ratio thus underlines the additional contraction brought by this binary search. For all the tested instances, the best value of ϵ falls between $\frac{1}{32}$ and $\frac{1}{8}$, which led us to fix ϵ to 10%. Further decreasing ϵ turns out to be a bad idea since the ratio remains quasi-constant (see Fig. 4.4–left in [1]).

The main contribution of this paper is an empirical analysis that has led us to an automatic tuning of the τ_{mohc} parameter during the search.

Table 1 contains the main results useful for our analysis. It reports the CPU time (first row of a multi-row) required by the solving strategy described above (based on `3BCID(Mohc)`) in function of τ_{mohc} . The NCSPs are sorted by decreasing gains in performance of `3BCID(Amohc)` w.r.t. `3BCID(HC4)` (column G/3B). The entries of columns 2 to 10 also contain (second row of a multi-row) a gain falling in $[0, 1]$. The gain is $\frac{\text{Time}(3BCID(\text{Mohc}))}{\text{Time}(3BCID(\text{Oracle}))}$, where `Oracle` is a theoretical `Mohc` variant that would be able to select the best value (yielding the results in bold in the table) for τ_{mohc} .

The first 10 columns highlight that the best value of τ_{mohc} nearly always falls in the range $[50\%, 99.99\%]$. The value 50% yields always better results than (or similar to) $\tau_{mohc} = 40\%$ or less [1]. Also, $\tau_{mohc} = 99.99\%$ is better than or similar to 100% (except in `ButcherA`). Note that `Mohc` with $\tau_{mohc} = 0\%$ is identical to `HC4` (see Algorithm 1).

Second, a significant correlation can be observed on Table 1 and on the graphic shown in Fig. 1. The curves show how the application frequency of the monotonicity-based procedures ($\frac{\text{number_of_calls}(\rho_{mohc}[f] \leq \tau_{mohc})}{\text{number_of_calls}(\text{Mohc-Revise})}$) evolves when τ_{mohc} increases. (Of course, the frequency becomes 1 when $\tau_{mohc} = 1$.) The correlation can be observed vertically for any value of τ_{mohc} , although this is clearer for intermediate values. For instance, with the abscissa $\tau_{mohc} = 60\%$,

¹ See www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html

² Compared to [2], the NCSP `kin1` has been removed because it can be solved with `3BCID(HC4)` in less than 100 choice points. `ButcherB` has been added. `ButcherA` (with $[a] = [-50, -1.1]$) and `ButcherB` (with $[a] = [-0.9, 50]$) are two sub-instances of the NCSP `Butcher`. Merlet has manually removed the value $a = 1$ to avoid a singularity...

Table 1. Experimental results. The column $G/3B = \frac{Time(3BCID(HC4))}{Time(3BCID(Amohc))}$ highlights the gain of our 3BCID(Amohc) contractor w.r.t. the standard 3BCID(HC4) (column 2).

NCSP		Mohc										Amohc G/3B		Amohc	
#var	#sol	τ_{mohc}	0%	40%	50%	60%	70%	80%	90%	99.99%	100%	policy 1	policy 2		
ButcherA	8 3	492480	67134	33560	16380	9317	6134	4071	3138	2949	605	680	3371	146	3988
		0.01	0.04	0.09	0.18	0.32	0.48	0.72	0.94	1	0.87				0.74
DirectKin	11 2	29882	2911	1479	1075	741	608	599	605	680	617	48.4	960		
		0.02	0.21	0.41	0.56	0.81	0.98	1	0.99	0.88	0.97		0.62		
Virasoro	8 224	12863	9268	5338	3050	1844	1660	1657	1544	1554	1552	8.29	2047		
		0.12	0.17	0.29	0.51	0.84	0.93	0.93	1	0.99	1		0.75		
ButcherB	8 3	3157	1752	1364	1100	914	705	579	471	479	459	6.87	474		
		0.15	0.27	0.35	0.43	0.52	0.67	0.81	1	0.98	1.03		1		
Geneig	6 10	655	421	293	219	181	161	150	143	163	143	4.60	140		
		0.22	0.34	0.49	0.65	0.79	0.89	0.95	1	0.88	1		1.02		
Yam.1-10	10 9	318	121	79.1	65.5	58.4	59.1	64.6	73.2	93.1	74.2	4.29	80.5		
		0.18	0.48	0.74	0.89	1	0.99	0.90	0.80	0.63	0.79		0.73		
Hayes	4 1	73.1	54.9	52.8	47.2	32.2	21.1	21.5	23.1	22.8	20.8	3.51	23.5		
		0.29	0.38	0.4	0.45	0.66	1	0.98	0.92	0.93	1.01		0.9		
Fourbar	4 3	1968	942	782	706	664	647	643	651	659	648	3.04	660		
		0.33	0.68	0.82	0.91	0.97	0.99	1	0.99	0.98	0.99		0.97		
Trigo1-10	10 9	214	146	97.4	84.2	95.1	94.1	114	123	115	108	1.99	80.0		
		0.39	0.58	0.86	1	0.89	0.89	0.74	0.69	0.73	0.78		1.05		
Pramanik	3 2	65.9	45.8	40.4	37.1	34.5	32.8	33.2	36.0	45.2	36.1	1.83	32.1		
		0.5	0.72	0.81	0.88	0.95	1	0.99	0.91	0.73	0.91		1.02		
Caprasse	4 18	3.91	3.84	3.59	3.72	3.74	3.84	3.26	3.4	3.33	3.52	1.11	3.72		
		0.83	0.85	0.91	0.88	0.87	0.85	1	0.96	0.98	0.92		0.88		
I5	10 30	87.0	82.9	80.0	77.1	75.6	74.6	73.6	79.0	180.2	79.5	1.09	78.0		
		0.85	0.89	0.92	0.95	0.97	0.99	1	0.93	0.41	0.93		0.94		
Brent-10	10 1008	33.3	33.3	32.7	33.0	35.6	44.9	50.6	73.5	72.7	32.7	1.02	52.5		
		0.98	0.98	1	0.99	0.92	0.73	0.65	0.44	0.45	1		0.62		
Trigexp2-11	11 0	139	136	136	136	139	136	140	182	225	136	1.02	193		
		0.98	1	1	1	0.98	1	0.97	0.75	0.61	1		0.71		
Eco9	9 16	22.1	21.6	21.5	21.7	22.1	23.5	27.5	41.3	46.6	22.3	0.99	23.0		
		0.98	1	1	0.99	0.98	0.92	0.78	0.52	0.46	0.97		0.94		
Redeco-8	8 8	9.58	9.58	9.61	9.75	9.89	10.9	12.2	16.9	19.5	9.78	0.98	10.2		
		1	1	1	0.98	0.97	0.88	0.78	0.57	0.49	0.98		0.94		
Katsura-12	12 7	64.7	65.7	64.3	73.0	87.9	168	249	232	234	85.5	0.76	69.2		
		0.99	0.98	1	0.88	0.73	0.38	0.26	0.28	0.28	0.75		0.93		
Average		0.52	0.62	0.71	0.77	0.83	0.86	0.85	0.80	0.73	0.94		0.87		

the application frequencies of Katsura, Redeco, Eco9, Trigexp2, Brent and I5 are inferior to 10%. It appears that these instances are not very well solved by 3BCID(Mohc) with $\tau_{mohc} = 100\%$ (compared to 3BCID(HC4)). This suggests that when the application frequency is low (resp. high), τ_{mohc} should rather be tuned to a low (resp. high) value. Intuitively, a high frequency means that a lot of monotonicity-based evaluations produce a sharp interval image, so that Mohc should well exploit these sharp evaluations.

This study has led us to a simple auto-adaptive τ_{mohc} tuning policy following three significant choices:

- Since Mohc-Revise exploits the monotonicity of a single function f , there is no reason that τ_{mohc} be the same for all constraints. This prevented the user

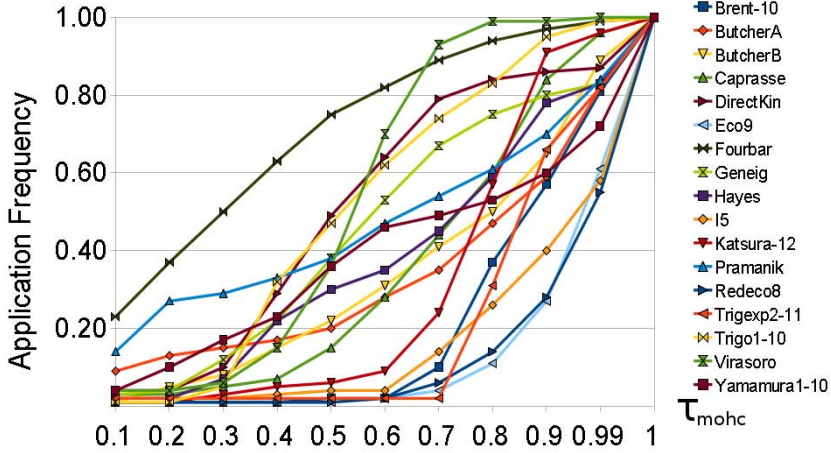


Fig. 1. Application frequency of the monot. Based procedures in function of τ_{mohc} .

from specifying a τ_{mohc} parameter for each function f , but this simplification is no more relevant in an adaptive tuning policy³.

- For any constraint f , $\tau_{mohc}[f]$ is fixed to one of both values 50% and 99.99%. Indeed, if we had an oracle able to select in any instance the best value for τ_{mohc} among 50% and 99.99%, the loss in performance w.r.t. an oracle knowing the best value of τ_{mohc} would be only 4%. $\tau_{mohc} = 50\%$ (resp. 99.99%) is generally the most relevant value when the gain in CPU time of 3BCID(Mohc) compared to 3BCID(HC4) is the smallest (resp. greatest).
- The application frequency is updated at each node of the search tree. It is more and more accurate as the number of measurements increases.

These choices lead to the tuning policy 1 based on Algorithm 2. The procedure `ComputeRhoMohc` is run for every constraint at each node of the search tree, before running the contraction procedures. `nb_calls[f]` and `nb_interesting[f]`, related to a given constraint f , are initialized to 0 before the search. The ratios `tau_mohc[f]` are set to 99.99% during the first 50 nodes before being adjusted in `ComputeRhoMohc`. The results are not sensitive to a fine tuning of `RHO_INTERESTING` and `TAU_FREQ`, empirically fixed to 65% and 10% respectively.

The performance of this policy is illustrated by the last line of Table 1. The loss in performance w.r.t. an oracle knowing the best value of τ_{mohc} (τ_{mohc} being common to all the constraints in a given NCSP) is 6% on average (at worst 25%; around 0% in 7 NCSPs). The obtained average gain (94%) highlights that `Amohc` is better than any fixed value of τ_{mohc} (i.e., 86% for $\tau_{mohc} = 80\%$). The column G/3B is also very convincing because 3BCID(`Amohc`) and 3BCID(`HC4`) have the same number of parameters, i.e., 0 coming from `HC4-Revise` and `Amohc-Revise`.

³ This explains why our auto-adaptive versions of `Mohc` sometimes (slightly) outperform the oracle so that the corresponding gains in the table are greater than 1.

Algorithm 2. ComputeRhoMohc (in f : a function, V : variables, $[V]$: domains)

```

nb_calls[f]++
rho_mohc[f] ←  $\frac{Diam([f]_M([V]))}{Diam([f]_N([V]))}$  /* rho_mohc[f] is computed */
if rho_mohc[f] < RHO_INTERESTING then nb_interesting[f]++; end if
interesting_freq ←  $\frac{nb\_interesting[f]}{nb\_calls[f]}$ 
if nb_calls[f] > 50 and interesting_freq < TAU_FREQ then
    tau_mohc[f] ← 50%
else
    tau_mohc[f] ← 99.99%
end if

```

An alternative policy 2 is based on contraction. Each time the monotonicity-based procedures of Algorithm 1 are applied, two ratios of box perimeters are computed: r_e is the gain in perimeter brought by HC4-Revise w.r.t. the initial box; r_m is the gain brought by MinMaxRevise+MonotonicBoxNarrow w.r.t. the previous box. If r_m is better (enough) than r_e , then τ_{mohc} is slightly incremented; if r_e is better than r_m , then it is decremented.

As shown in Table [11](#), the policy 2 is not so bad although less efficient than the policy 1 on average (87%). Unfortunately, this adaptive policy is the worst when Mohc is the most useful (e.g., see ButcherA, DirectKin, Virasoro).

References

1. Araya, I.: Exploiting Common Subexpressions and Monotonicity of Functions for Filtering Algorithms over Intervals. PhD thesis, University of Nice-Sophia (2010)
2. Araya, I., Trombettoni, G., Neveu, B.: Exploiting Monotonicity in Interval Constraint Propagation. In: Proc. AAAI (to appear, 2010)
3. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.-F.: Revising Hull and Box Consistency. In: Proc. ICLP, pp. 230–244 (1999)
4. Chabert, G.: Ibex – An Interval Based EXplorer (2010), <http://www.ibex-lib.org>
5. Collavizza, H., Delobel, F., Rueher, M.: Comparing Partial Consistencies. Reliable Comp. 5(3), 213–228 (1999)
6. Jaulin, L.: Interval Constraint Propagation with Application to Bounded-error Estimation. Automatica 36, 1547–1552 (2000)
7. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht (1996)
8. Kieffer, M., Jaulin, L., Walter, E., Meizel, D.: Robust Autonomous Robot Localization Using Interval Analysis. Reliable Computing 3(6), 337–361 (2000)
9. Lhomme, O.: Consistency Tech. for Numeric CSPs. In: IJCAI, pp. 232–238 (1993)
10. Merlet, J.-P.: Interval Analysis and Robotics. In: Symp. of Robotics Research (2007)
11. Moore, R.E.: Interval Analysis. Prentice-Hall, Englewood Cliffs (1966)
12. Rueher, M., Goldsztejn, A., Lebbah, Y., Michel, C.: Capabilities of Constraint Programming in Rigorous Global Optimization. In: NOLTA (2008)
13. Trombettoni, G., Chabert, G.: Constructive Interval Disjunction. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 635–650. Springer, Heidelberg (2007)
14. Van Hentenryck, P., Michel, L., Deville, Y.: Numerica: A Modeling Language for Global Optimization. MIT Press, Cambridge (1997)

Improving the Performance of maxRPC

Thanasis Balafoutis¹, Anastasia Paparrizou², Kostas Stergiou^{1,2,*}, and Toby Walsh³

¹ Department of Information and Communication Systems Engineering,
University of the Aegean, Greece

² Department of Informatics and Telecommunications Engineering,
University of Western Macedonia, Greece

³ NICTA, University of New South Wales, Australia

Abstract. Max Restricted Path Consistency (maxRPC) is a local consistency for binary constraints that can achieve considerably stronger pruning than arc consistency. However, existing maxRPC algorithms suffer from overheads and redundancies as they can repeatedly perform many constraint checks without triggering any value deletions. In this paper we propose techniques that can boost the performance of maxRPC algorithms. These include the combined use of two data structures to avoid many redundant constraint checks, and heuristics for the efficient ordering and execution of certain operations. Based on these, we propose two closely related maxRPC algorithms. The first one has optimal $O(end^3)$ time complexity, displays good performance when used stand-alone, but is expensive to apply during search. The second one has $O(en^2d^4)$ time complexity, but a restricted version with $O(end^4)$ complexity can be very efficient when used during search. Both algorithms have $O(ed)$ space complexity when used stand-alone. However, the first algorithm has $O(end)$ space complexity when used during search, while the second retains the $O(ed)$ complexity. Experimental results demonstrate that the resulting methods constantly outperform previous algorithms for maxRPC, often by large margins, and constitute a more than viable alternative to arc consistency.

1 Introduction

maxRPC is a strong domain filtering consistency for binary constraints introduced in 1997 by Debruyne and Bessiere [5]. maxRPC achieves a stronger level of local consistency than arc consistency (AC), and in [6] it was identified, along with singleton AC (SAC), as a promising alternative to AC. Although SAC has received considerable attention since, maxRPC has been comparatively overlooked. The basic idea of maxRPC is to delete any value a of a variable x that has no arc consistency (AC) or path consistency (PC) support in a variable y . A value b is an AC support for a if the two values are compatible, and it is also a PC support for a if this pair of values is path consistent. A pair of values (a, b) is path consistent iff for every third variable there exists at least one value, called a PC witness, that is compatible with both a and b .

The first algorithm for maxRPC was proposed in [5], and two more algorithms have been proposed since then [7][10]. The algorithms of [5] and [10] have been evaluated on

* Part of this work was carried out when the 3rd author was at NICTA, Australia.

random problems only, while the algorithm of [7] has not been experimentally evaluated at all. Despite achieving considerable pruning, existing maxRRC algorithms suffer from overhead and redundancies as they can repeatedly perform many constraint checks without triggering any value deletions. These constraint checks occur when a maxRPC algorithm searches for an AC support for a value and when, having located one, it checks if it is also a PC support by looking for PC witnesses in other variables. As a result, the use of maxRRC during search often slows down the search process considerably compared to AC, despite the savings in search tree size.

In this paper we propose techniques to improve the applicability of maxRPC by eliminating some of these redundancies while keeping a low space complexity. We also investigate approximations of maxRPC that only make slightly fewer value deletions in practice, while being significantly faster. We first demonstrate that we can avoid many redundant constraint checks and speed up the search for AC and PC supports through the careful and combined application of two data structures already used by maxRPC and AC algorithms [7,10,2,8,9]. Based on this, we propose a coarse-grained maxRPC algorithm called maxRPC3 with optimal $O(end^3)$ time complexity. This algorithm displays good performance when used stand-alone (e.g. for preprocessing), but is expensive to apply during search. We then propose another maxRPC algorithm, called maxRPC3^{rm}. This algorithm has $O(en^2d^4)$ time complexity, but a restricted version with $O(end^4)$ complexity can be very efficient when used during search through the use of *residues*. Both algorithms have $O(ed)$ space complexity when used stand-alone. However, maxRPC3 has $O(end)$ space complexity when used during search, while maxRPC3^{rm} retains the $O(ed)$ complexity.

Similar algorithmic improvements can be applied to *light maxRPC* (lmaxRPC), an approximation of maxRPC [10]. This achieves a lesser level of consistency compared to maxRPC but still stronger than AC, and is more cost-effective than maxRPC when used during search. Experiments confirm that lmaxRPC is indeed a considerably better option than maxRPC.

We also propose a number of heuristics that can be used to efficiently order the searches for PC supports and witnesses. Interestingly, some of the proposed heuristics not only reduce the number of constraint checks but also the number of visited nodes.

We make a detailed experimental evaluation of new and existing algorithms on various problem classes. This is the first wide experimental study of algorithms for maxRPC and its approximations on benchmark non-random problems. Results show that our methods constantly outperform existing algorithms, often by large margins. When applied during search our best method offers up to one order of magnitude reduction in constraint checks, while cpu times are improved up to four times compared to the best existing algorithm. In addition, these speed-ups enable a search algorithm that applies lmaxRPC to compete with or outperform MAC on many problems.

2 Background and Related Work

A *Constraint Satisfaction Problem* (CSP) is defined as a tuple (X, D, C) where: $X = \{x_1, \dots, x_n\}$ is a set of n variables, $D = \{D(x_1), \dots, D(x_n)\}$ is a set of domains, one for each variable, with maximum cardinality d , and $C = \{c_1, \dots, c_e\}$ is a set of e

constraints. Each constraint c is a pair $(var(c), rel(c))$, where $var(c) = \{x_1, \dots, x_m\}$ is an ordered subset of X , and $rel(c)$ is a subset of the Cartesian product $D(x_1) \times \dots \times D(x_m)$ that specifies the allowed combinations of values for the variables in $var(c)$. In the following, a binary constraint c with $var(c) = \{x_i, x_j\}$ will be denoted by c_{ij} , and $D(x_i)$ will denote the current domain of variable x_i . Each tuple $\tau \in rel(c)$ is an ordered list of values (a_1, \dots, a_m) such that $a_j \in D(x_j), j = 1, \dots, m$. A tuple $\tau \in rel(c_i)$ is *valid* iff none of the values in the tuple has been removed from the domain of the corresponding variable.

The process which verifies whether a given tuple is allowed by a constraint c is called a *constraint check*. A binary CSP is a CSP where each constraint involves at most two variables. We assume that binary constraint checks are performed in constant time. In a binary CSP, a value $a_i \in D(x_i)$ is *arc consistent* (AC) iff for every constraint c_{ij} there exists a value $a_j \in D(x_j)$ s.t. the pair of values (a_i, a_j) satisfies c_{ij} . In this case a_j is called an *AC-support* of a_i . A variable is AC iff all its values are AC. A problem is AC iff there is no empty domain in D and all the variables in X are AC.

2.1 maxRPC

A value $a_i \in D(x_i)$ is *max restricted path consistent* (maxRPC) iff it is AC and for each constraint c_{ij} there exists a value $a_j \in D(x_j)$ that is an AC-support of a_i s.t. the pair of values (a_i, a_j) is *path consistent* (PC) [5]. A pair of values (a_i, a_j) is PC iff for any third variable x_k there exists a value $a_k \in D(x_k)$ s.t. a_k is an AC-support of both a_i and a_j . In this case a_j is a *PC-support* of a_i in x_j and a_k is a *PC-witness* for the pair (a_i, a_j) in x_k . A variable is maxRPC iff all its values are maxRPC. A problem is maxRPC iff there is no empty domain and all variables are maxRPC.

To our knowledge, three algorithms for achieving maxRPC have been proposed in the literature so far. The first one, called maxRPC1, is a fine-grained algorithm based on AC6 and has optimal $O(end^3)$ time complexity and $O(end)$ space complexity [5]. The second algorithm, called maxRPC2, is a coarse-grained algorithm having $O(end^3)$ time and $O(ed)$ space complexity [7]. Finally, maxRPC^{rm} is a coarse-grained algorithm based on AC3^{rm} [10]. The time and space complexities of maxRPC^{rm} are $O(en^2d^4)$ and $O(end)$ but it has some advantages compared to the other two because of its lighter use of data structures. Among the three algorithms maxRPC2 seems to be the most promising for stand-alone use as it has a better time and space complexity than maxRPC^{rm} without requiring heavy data structures or complex implementation as maxRPC1 does. On the other hand, maxRPC^{rm} can be better suited for use during search as it avoids the costly maintenance of data structures.

Central to maxRPC2 is the *LastPC* data structure, as we call it here. For each constraint c_{ij} and each value $a_i \in D(x_i)$, $LastPC_{x_i, a_i, x_j}$ gives the most recently discovered PC-support of a_i in $D(x_j)$. maxRPC2 maintains this data structure incrementally. This means that the data structure is copied when moving forward during search (i.e. after a successfully propagated variable assignment) and restored when backtracking (after a failed variable assignment). This results in the following behavior: When looking for a PC-support for a_i in $D(x_j)$, maxRPC2 first checks if $LastPC_{x_i, a_i, x_j}$ is valid. If it is not, it searches for a new PC-support starting from the value immediately after $LastPC_{x_i, a_i, x_j}$ in $D(x_j)$. In this way a good time complexity bound is achieved. On

the other hand, maxRPC^{rm} uses a data structure similar to *LastPC* to store *residues*, i.e. supports that have been discovered during execution and stored for future use, but does not maintain this structure incrementally¹. When looking for a PC-support for a_i in $D(x_j)$, if the residue $\text{LastPC}_{x_i, a_i, x_j}$ is not valid then maxRPC^{rm} searches for a new PC-support from scratch in $D(x_j)$. This results in higher complexity, but crucially does not require costly maintenance of the *LastPC* data structure during search.

A major overhead of both maxRPC2 and maxRPC^{rm} is the following. When searching for a PC-witness for a pair of values (a_i, a_j) in a third variable x_k , they always start the search from scratch, i.e. from the first available value in $D(x_k)$. As these searches can be repeated many times during search, there can be many redundant constraint checks. In contrast, maxRPC1 manages to avoid searching from scratch through the use of an additional data structure. This saves many constraint checks, albeit resulting in $O(\text{end})$ space complexity and requiring costly maintenance of this data structure during search. The algorithms we describe below largely eliminate these redundant constraint checks with lower space complexity, and in the case of maxRPC3^{rm} with only light use of data structures.

3 New Algorithms for maxRPC

We first recall the basic ideas of algorithms maxRPC2 and maxRPC^{rm} as described in [7] and [10]. Both algorithms use a propagation list L where variables whose domain is pruned are added. Once a variable x_j is removed from L all neighboring variables are revised to delete any values that are no longer maxRPC. For any value a_i of such a variable x_i there are two possible reasons for deletion. The first, which we call *PC-support loss* hereafter, is when the unique PC-support $a_j \in D(x_j)$ for a_i has been deleted. The second, which we call *PC-witness loss* hereafter, is when the unique PC-witness $a_j \in D(x_j)$ for the pair (a_i, a_k) , where a_k is the unique PC-support for a_i on some variable x_k , has been deleted. In both cases value a_i is no longer maxRPC.

We now give a unified description of algorithms maxRPC3 and maxRPC3^{rm} . Both algorithms utilize data structures *LastPC* and *LastAC* which have the following functionalities: For each constraint c_{ij} and each value $a_i \in D(x_i)$, $\text{LastPC}_{x_i, a_i, x_j}$ and $\text{LastAC}_{x_i, a_i, x_j}$ give (point to) the most recently discovered PC and AC supports of a_i in $D(x_j)$ respectively. Initially, all *LastPC* and *LastAC* pointers are set to a special value NIL, considered to precede all values in any domain. Algorithm maxRPC3 updates the *LastPC* and *LastAC* structures incrementally like maxRPC2 and AC2001/3.1 respectively do. In contrast, algorithm maxRPC3^{rm} uses these structures as residues like maxRPC^{rm} and AC^{rm} do.

The pseudocode for the unified description of maxRPC3 and maxRPC3^{rm} is given in Algorithm 1 and Functions 2, 3, 4. We assume the existence of a global Boolean variable RM which determines whether the algorithm presented is instantiated to maxRPC3 or to maxRPC3^{rm} . If RM is true, the algorithm used is maxRPC3^{rm} . Otherwise, the algorithm is maxRPC3 .

Being coarse-grained, Algorithm 1 uses a propagation list L where variables that have their domain filtered are inserted. If the algorithm is used for preprocessing then,

¹ maxRPC^{rm} also uses residues in a different context.

during an initialization phase, for each value a_i of each variable x_i we check if a_i is maxRPC. If it is not then it is deleted from $D(x_i)$ and x_i is added to L . The initialization function is not shown in detail due to limited space. If the algorithm is used during search then L is initialized with the currently assigned variable (line 3).

In the main part of Algorithm [1](#), when a variable x_j is removed from L , each variable x_i constrained with x_j must be made maxRPC. For each value $a_i \in D(x_i)$ Algorithm [1](#) like maxRPC2 and maxRPC^{rm}, establishes if a_i is maxRPC by checking for PC-support loss and PC-witness loss at lines 8 and 12.

Algorithm 1. maxRPC3 / maxRPC3^{rm}

```

1: if preprocessing then
2:   if  $\neg$ initialization(L, LastPC, LastAC) then return FAILURE;
3: else L = {currently assigned variable};
4: while L  $\neq$   $\emptyset$  do
5:   L=L- $\{x_j\}$ ;
6:   for each  $x_i \in X$  s.t.  $c_{ij} \in C$  do
7:     for each  $a_i \in D(x_i)$  do
8:       if  $\neg$ searchPCsup( $a_i, x_j$ ) then
9:         delete  $a_i$ ;
10:        L=L  $\cup$   $\{x_i\}$ ;
11:      else
12:        if  $\neg$ checkPCwit( $a_i, x_j$ ) then
13:          delete  $a_i$ ;
14:          L=L  $\cup$   $\{x_i\}$ ;
15:        if  $D(x_i)$  is empty then return FAILURE;
16: return SUCCESS;
```

First, function *searchPCsup* is called to check if a PC-support for a_i exists in $D(x_j)$. If value $LastPC_{x_i, a_i, x_j}$ is still in $D(x_j)$, then *searchPCsup* returns TRUE (lines 1-2). If $LastPC_{x_i, a_i, x_j}$ is not valid, we search for a new PC-support. If maxRPC3 is used, we can take advantage of the *LastPC* and *LastAC* pointers to avoid starting this search from scratch. Specifically, we know that no PC-support can exist before $LastPC_{x_i, a_i, x_j}$, and also none can exist before $LastAC_{x_i, a_i, x_j}$, since all values before $LastAC_{x_i, a_i, x_j}$ are not AC-supports of a_i . Lines 5-6 in *searchPCsup* take advantage of these to locate the appropriate starting value b_j . Note that maxRPC2 always starts the search for a PC-support from the value after $LastPC_{x_i, a_i, x_j}$. If the algorithm is called during search, in which case we use maxRPC3^{rm} then the search for a new PC-support starts from scratch (line 8), just like maxRPC^{rm} does.

For every value $a_j \in D(x_j)$, starting with b_j , we first check if is an AC-support of a_i (line 10). This is done using function *isConsistent* which simple checks if two values are compatible. If it is, and the algorithm is maxRPC3, then we can update $LastAC_{x_i, a_i, x_j}$ under a certain condition (lines 12-13). Specifically, if $LastAC_{x_i, a_i, x_j}$ was deleted from $D(x_j)$, then we can set $LastAC_{x_i, a_i, x_j}$ to a_j in case $LastAC_{x_i, a_i, x_j} > LastPC_{x_i, a_i, x_j}$. If $LastAC_{x_i, a_i, x_j} \leq LastPC_{x_i, a_i, x_j}$ then we cannot do this as there may be AC-supports for a_i between $LastAC_{x_i, a_i, x_j}$ and $LastPC_{x_i, a_i, x_j}$ in the lexicographical ordering. We then move on to verify the path consistency of (a_i, a_j) through function *searchPCwit*.

Function 2. *searchPCsup*(a_i, x_j):boolean

```

1: if  $LastPC_{x_i, a_i, x_j} \in D(x_j)$  then
2:   return true;
3: else
4:   if  $\neg RM$  then
5:     if  $LastAC_{x_i, a_i, x_j} \in D(x_j)$  then  $b_j = \max>LastPC_{x_i, a_i, x_j} + 1, LastAC_{x_i, a_i, x_j}$ ;
6:     else  $b_j = \max>LastPC_{x_i, a_i, x_j} + 1, LastAC_{x_i, a_i, x_j} + 1$ ;
7:   else
8:      $b_j =$  first value in  $D(x_j)$ ;
9:   for each  $a_j \in D(x_j), a_j \geq b_j$  do
10:    if isConsistent( $a_i, a_j$ ) then
11:      if  $\neg RM$  then
12:        if  $LastAC_{x_i, a_i, x_j} \notin D(x_j)$  AND  $LastAC_{x_i, a_i, x_j} > LastPC_{x_i, a_i, x_j}$  then
13:           $LastAC_{x_i, a_i, x_j} = a_j$ ;
14:        if searchPCwit( $a_i, a_j$ ) then
15:           $LastPC_{x_i, a_i, x_j} = a_j$ ;
16:        if RM then  $LastAC_{x_i, a_i, x_j} = a_j; LastPC_{x_j, a_j, x_i} = a_i$ ;
17:        return true;
18: return false;
```

If no PC-support for a_i is found in $D(x_j)$, *searchPCsup* will return FALSE, a_i will be deleted and x_i will be added to L . Otherwise, $LastPC_{x_i, a_i, x_j}$ is set to the discovered PC-support a_j (line 15). If $\maxRPC3^{rm}$ is used then we update the residue $LastAC_{x_i, a_i, x_j}$ since the discovered PC-support is also an AC-support. In addition, to exploit the multidirectionality of residues, $\maxRPC3^{rm}$ sets $LastPC_{x_j, a_j, x_i}$ to a_i , as in [10].

Function *searchPCwit* checks if a pair of values (a_i, a_j) is PC by doing the following for each variable x_k constrained with x_i and x_j .² First, it checks if either $LastAC_{x_i, a_i, x_k}$ is valid and consistent with a_j or $LastAC_{x_j, a_j, x_k}$ is valid and consistent with a_i (line 3). If one of these conditions holds then we have found a PC-witness for (a_i, a_j) without searching in $D(x_k)$ and we move on to the next variable constrained with x_i and x_j . Note that neither $\maxRPC2$ nor \maxRPC^{rm} can do this as they do not have the *LastAC* structure. Experimental results in Section 5 demonstrate that these simple conditions can eliminate a very large number of redundant constraint checks.

Function 3. *searchPCwit*(a_i, a_j):boolean

```

1: for each  $x_k \in V$  s.t.  $c_{ik} \in C$  and  $c_{jk} \in C$  do
2:    $\maxRPCsupport = FALSE$ ;
3:   if  $(LastAC_{x_i, a_i, x_k} \in D(x_k)$  AND isConsistent( $LastAC_{x_i, a_i, x_k}, a_j$ )) OR  $(LastAC_{x_j, a_j, x_k} \in D(x_k)$  AND isConsistent( $LastAC_{x_j, a_j, x_k}, a_i$ )) then continue;
4:   if  $\neg RM$  then
5:     if  $\neg seekACsupport(x_i, a_i, x_k)$  OR  $\neg seekACsupport(x_j, a_j, x_k)$  then return false;
6:      $b_k = \max>LastAC_{x_i, a_i, x_k}, LastAC_{x_j, a_j, x_k}$ ;
7:   else  $b_k =$  first value in  $D(x_k)$ ;
8:   for each  $a_k \in D(x_k), a_k \geq b_k$  do
9:     if isConsistent( $a_i, a_k$ ) AND isConsistent( $a_j, a_k$ ) then
10:      if RM then  $LastAC_{x_i, a_i, x_k} = LastAC_{x_j, a_j, x_k} = a_k$ ;
11:       $\maxRPCsupport = TRUE$ ; break;
12:   if  $\neg \maxRPCsupport$  then return false;
13: return true;
```

² Since AC is enforced by the \maxRPC algorithm, we only need to consider variables that form a 3-clique with x_i and x_j .

If none of the conditions holds then we have to search in $D(x_k)$ for a PC-witness. If the algorithm is maxRPC3 then we can exploit the $LastAC$ structure to start this search from $b_k = \max\{LastAC_{x_i, a_i, x_k}, LastAC_{x_j, a_j, x_k}\}$ (line 6). But before doing this, we call function $seekACsupport$ (not shown for space reasons), first with (x_i, a_i, x_k) and then with (x_j, a_j, x_k) as parameters, to find the lexicographically smallest AC-supports for a_i and a_j in $D(x_k)$ (line 5). If such supports are found, $LastAC_{x_i, a_i, x_k}$ and $LastAC_{x_j, a_j, x_k}$ are updated accordingly. In case no AC-support is found for either a_i or a_j then $seekACsupport$ returns FALSE, and subsequently $searchPCwit()$ will also return FALSE.

If the algorithm used is maxRPC3^{rm} then we start search for a PC-witness from scratch (line 7), as maxRPC2 and maxRPC^{rm} always do. If a PC-witness a_k is found (line 9) and we are using maxRPC3^{rm} then both residues $LastAC_{x_i, a_i, x_k}$ and $LastAC_{x_j, a_j, x_k}$ are set to a_k as they are the most recently discovered AC-supports. If no PC-witness is found then we have determined that the pair (a_i, a_j) is not PC and as a result FALSE will be returned and $searchPCsup$ will move to check if the next available value in $D(x_j)$ is a PC-support for a_i .

Function 4. $checkPCwit(a_i, x_j)$:boolean

```

1: for each  $x_k \in V$  s.t.  $c_{ik} \in C$  and  $c_{kj} \in C$  do
2:   witness=FALSE; findPCsupport=FALSE;
3:   if  $a_k = LastPC_{x_i, a_i, x_k} \in D(x_k)$  then
4:     if  $(LastAC_{x_i, a_i, x_j} \in D(x_j) \text{ AND } isConsistent(LastAC_{x_i, a_i, x_j}, a_k)) \text{ OR } (LastAC_{x_k, a_k, x_j} \in D(x_j) \text{ AND } isConsistent(LastAC_{x_k, a_k, x_j}, a_i))$  then
5:       witness=TRUE;
6:     else
7:       if  $\neg RM$  then
8:         if  $seekACsupport(x_i, a_i, x_j)$  AND  $seekACsupport(x_k, a_k, x_j)$  then
9:            $b_j = \max(LastAC_{x_i, a_i, x_j}, LastAC_{x_k, a_k, x_j})$ ;
10:          else findPCsupport=TRUE;
11:          else  $b_j = \text{first value in } D(x_j)$ ;
12:          if  $\neg findPCsupport$  then
13:            for each  $a_j \in D(x_j), a_j \geq b_j$  do
14:              if  $isConsistent(a_i, a_j)$  AND  $isConsistent(a_k, a_j)$  then
15:                if RM then  $LastAC_{x_i, a_i, x_j} = LastAC_{x_k, a_k, x_j} = a_j$ ;
16:                witness=TRUE; break;
17:          if  $\neg$ witness AND exists  $a_k > LastPC_{x_i, a_i, x_k} \in D(x_k)$  then
18:            if  $\neg RM$  then
19:              if  $LastAC_{x_i, a_i, x_k} \in D(x_k)$  then  $b_k = \max(LastPC_{x_i, a_i, x_k} + 1, LastAC_{x_i, a_i, x_k})$ ;
20:              else  $b_k = \max(LastPC_{x_i, a_i, x_k} + 1, LastAC_{x_i, a_i, x_k} + 1)$ ;
21:            else
22:               $b_k = \text{first value in } D(x_k)$ ;
23:            for each  $a_k \in D(x_k), a_k \geq b_k$  do
24:              if  $isConsistent(a_i, a_k)$  then
25:                if  $\neg RM$  then
26:                  if  $LastAC_{x_i, a_i, x_k} \notin D(x_k)$  AND  $LastAC_{x_i, a_i, x_k} > LastPC_{x_i, a_i, x_k}$  then
27:                     $LastAC_{x_i, a_i, x_k} = a_k$ ;
28:                  if  $searchPCwit(a_i, a_k)$  then
29:                     $LastPC_{x_i, a_i, x_k} = a_k$ ;
30:                  if RM then  $LastAC_{x_i, a_i, x_k} = a_k; LastPC_{x_k, a_k, x_i} = a_i$ ;
31:                  witness=TRUE; break;
32:              if  $\neg$ witness then return false;
33: return true;
```

If value a_i is not removed by *searchPCsup* in Algorithm [11](#) function *checkPCwit* is called to check for PC-witness loss. This is done by iterating over the variables that are constrained with both x_i and x_j . For each such variable x_k , we first check if $a_k = LastPC_{x_i, a_i, x_k}$ is still in $D(x_k)$ (line 3). If so then we check if there still is a PC-witness in $D(x_j)$. This is done by first checking if either $LastAC_{x_i, a_i, x_j}$ is valid and consistent with a_k or $LastAC_{x_k, a_k, x_j}$ is valid and consistent with a_i (line 4). If neither of these conditions holds then we search for a PC-witness starting from $b_j = \max\{LastAC_{x_i, a_i, x_j}, LastAC_{x_k, a_k, x_j}\}$ in case of $\maxRPC3$ (line 9), after checking the existence of AC-supports for a_i and a_k in $D(x_j)$, by calling *seekACsupport* (line 8). If there is no AC-support in $D(x_j)$ for either a_i or a_k we set the auxiliary Boolean variable *findPCsupport* to TRUE to avoid searching for a PC-witness.

If $\maxRPC3^{rm}$ is used, we start searching for a PC-witness from scratch (line 11). Note that $\maxRPC2$ does not do the check of line 4 and always starts the search for a PC-witness from the first value in $D(x_j)$. In contrast, \maxRPC^{rm} avoids some redundant checks through the use of special residues, albeit resulting in $O(end)$ space complexity. When using $\maxRPC3^{rm}$, for each value $a_j \in D(x_j)$ we check if it is compatible with a_i and a_k and move the *LastAC* pointers accordingly (lines 14-15), exploiting the multidirectionality of residues,

If $LastPC_{x_i, a_i, x_k}$ has been removed or a_i has no PC-witness in $D(x_j)$, we search for a new PC-support for a_i in $D(x_k)$. As in function *searchPCsup*, when $\maxRPC3$ is used this search starts at an appropriate value calculated taking advantage of $LastPC_{x_i, a_i, x_k}$ and $LastAC_{x_i, a_i, x_k}$ (lines 18-20). When $\maxRPC3^{rm}$ is used we start from scratch. If an AC-support for a_i is found (line 24), we check if it is also a PC-support by calling function *searchPCwit* (line 28). If $\maxRPC3$ is used then $LastAC_{x_i, a_i, x_k}$ is updated when necessary (lines 26-27). If a PC-support is found, $LastPC_{x_i, a_i, x_k}$ is set accordingly (line 29). If $\maxRPC3^{rm}$ is used then the residue $LastAC_{x_i, a_i, x_k}$ is also updated, as is $LastPC_{x_k, a_k, x_i}$ (bidirectionality). If the search for a PC-support fails then FALSE will be returned, a_i will be deleted, and x_i will be added to L.

3.1 Light maxRPC

Light maxRPC (lmaxRPC) is an approximation of maxRPC that only propagates the loss of AC-supports and not the loss of PC-witnesses [\[10\]](#). This ensures that the obtained algorithm enforces a consistency property that is at least as strong as AC.

lmaxRPC is a procedurally defined local consistency, meaning that its description is tied to a specific maxRPC algorithm. Light versions of algorithms $\maxRPC3$ and $\maxRPC3^{rm}$, simply noted $l\maxRPC3$ and $l\maxRPC3^{rm}$ respectively, can be obtained by omitting the call to the *checkPCwit* function (lines 11-14 of Algorithm [11](#)). In a similar way, we can obtain light versions of algorithms $\maxRPC2$ and \maxRPC^{rm} .

As already noted in [\[10\]](#), the light versions of different maxRPC algorithms may not be equivalent in terms of the pruning they achieve. To give an example, a brute force algorithm for lmaxRPC that does not use any data structures can achieve more pruning than algorithms $l\maxRPC2$, $l\maxRPC3$, and $l\maxRPC^{rm}$, albeit being much slower in practice. Consider that any of these three algorithms will return TRUE in case $LastPC_{x_i, a_i, x_j}$ is valid. However, although $LastPC_{x_i, a_i, x_j}$ is valid, it may no longer

be a PC-support because the PC-witness in some third variable may have been deleted, and it may be the last one. In a case where $LastPC_{x_i, a_i, x_j}$ was the last PC-support in x_j for value a_i , the three advanced algorithms will not delete a_i while the brute force one will. This is because it will exhaustively check all values of x_j for PC-support, concluding that there is none.

The worst-case time and space complexities of algorithm $lmaxRPC2$ are the same as $maxRPC2$. Algorithm $lmaxRPC^{rm}$ has $O(n^3d^4)$ time and $O(ed)$ space complexities, which are lower than those of $maxRPC^{rm}$. Experiments with random problems using algorithms $lmaxRPC^{rm}$ and $maxRPC^{rm}$ showed that the pruning power of $lmaxRPC$ is only slightly weaker than that of $maxRPC$ [10]. At the same time, it can offer significant gains in run times when used during search. These results were also verified by us through a series of experiments on various problem classes.

3.2 Complexities

We now discuss the complexities of algorithms $maxRPC3$ and $maxRPC3^{rm}$ and their light versions. To directly compare with existing algorithms for (1)maxRPC, the time complexities give the asymptotic number of constraint checks³. Following [9], the *node* time (resp. space) complexity of a (1)maxRPC algorithm is the worst-case time (resp. space) complexity of invoking the algorithm after a variable assignment. The corresponding *branch* complexities of an (1)maxRPC algorithm are the worst-case complexities of any incremental sequence of $k \leq n$ invocations of the algorithm. That is, the complexities of incrementally running the algorithm down a branch of the search tree until a fail occurs.

Proposition 1. The node and branch time complexity of (1)maxRPC3 is $O(end^3)$.

Proof. The complexity is determined by the total number of calls to function *isConsistent* in *searchPCsup*, *checkPCwit*, and mainly *searchPCwit* where most checks are executed.

Each variable can be inserted and extracted from L every time a value is deleted from its domain, giving $O(d)$ times in the worst case. Each time a variable x_j is extracted from L , *searchPCsup* will look for a PC-support in $D(x_j)$ for all values $a_i \in D(x_i)$, s.t. $c_{ij} \in C$. For each variable x_i , $O(d)$ values are checked. Checking if a value $a_j \in D(x_j)$ is a PC-support involves first checking in $O(1)$ if it is an AC-support (line 10 in Function 2) and then calling *searchPCwit*. The cost of *searchPCwit* is $O(n + nd)$ since there are $O(n)$ variables constrained with both x_i and x_j and, after making the checks in line 3, their domains must be searched for a PC-witness, each time from scratch with cost $O(nd)$. Through the use of *LastPC* no value of x_j will be checked more than once over all the $O(d)$ times x_j is extracted from L , meaning that for any value $a_i \in D(x_i)$ and any variable x_j , the overall cost of *searchPCwit* will be $O(dn + nd^2) = O(nd^2)$. Hence, *searchPCsup* will cost $O(nd^2)$ for one value of x_i , giving $O(nd^3)$ for d values. Since, in the worst case, this process will be repeated for every pair of variables x_i and x_j that are constrained, the total cost of *searchPCsup* will be $O(end^3)$. This is the node complexity of $lmaxRPC3$.

³ However, constraint checks do not always reflect run times as other operations may have an equal or even greater effect.

In *checkPCwit* the algorithms iterate over the variables in a triangle with x_j and x_i . In the worst case, for each such variable x_k , $D(x_j)$ will be searched from scratch for a PC-witness of a_i and its current PC-support in x_k . As x_j can be extracted from L $O(d)$ times and each search from scratch costs $O(d)$, the total cost of checking for a PC-witness in $D(x_j)$, including the checks of line 4 in Function [4](#), will be $O(d + d^2)$. For d values of x_i this will be $O(d^3)$. As this process will be repeated for all triangles of variables, whose number is bounded by en , its total cost will be $O(end^3)$. If no PC-witness is found then a new PC-support for a_i in $D(x_k)$ is sought through *searchPCwit*. This costs $O(nd^2)$ as explained above but it is amortized with the cost incurred by the calls to *searchPCwit* from *searchPCsup*. Therefore, the cost of *checkPCwit* is $O(end^3)$. This is also the node complexity of $\max\text{RPC3}$.

The branch complexity of $(1)\max\text{RPC3}$ is also $O(end^3)$. This is because the use of *LastPC* ensures that for any constraint c_{ij} and a value $a_i \in D(x_i)$, each value of x_j will be checked at most once for PC-support while going down the branch. Therefore, the cost of *searchPCwit* is amortized. \square

Proposition 2. The node and branch time complexities of $1\max\text{RPC3}^{rm}$ and $\max\text{RPC3}^{rm}$ are $O(end^4)$ and $O(en^2d^4)$ respectively.

Proof. The proof is not given in detail due to lack of space. The main difference with $1\max\text{RPC3}$ is that since *lastPC* is not updated incrementally, each time we seek a PC-support for a value $a_i \in D(x_i)$ in x_j , $D(x_j)$ will be searched from scratch in the worst case. This incurs an extra $O(d)$ cost to *searchPCsup* and *searchPCwit*. Hence, the node complexity of $1\max\text{RPC3}^{rm}$ is $O(end^4)$. Also, the total cost of *searchPCwit* in one node cannot be amortized. This means that the cost of *searchPCwit* within *checkPCwit* is $O(nd^2)$. Hence, the node complexity of $\max\text{RPC3}^{rm}$ is $O(en^2d^4)$. The branch complexities are the same because the calls to *searchPCwit* are amortized. \square

The space complexities of the algorithms are determined by the space required for data structures *LastPC* and *LastAC*. Since both require $O(ed)$ space, this is the node space complexity of $(1)\max\text{RPC3}$ and $(1)\max\text{RPC3}^{rm}$. $(1)\max\text{RPC3}$ has $O(end)$ branch space complexity because of the extra space required for the incremental update and restoration of the data structures. As $(1)\max\text{RPC3}^{rm}$ avoid this, its branch space complexity is $O(ed)$.

4 Heuristics for maxRPC Algorithms

Numerous heuristics for ordering constraint or variable revisions have been proposed and used within AC algorithms [[11](#)][[13](#)][[11](#)]. Heuristics such as the ones used by AC algorithms can be also used within a maxRPC algorithm to efficiently select the next variable to be removed from the propagation list (line 5 of Algorithm [11](#)). In addition to this, maxRPC and $1\max\text{RPC}$ algorithms can benefit from the use of heuristics elsewhere in their execution. Once a variable x_j has been removed from the propagation list, heuristics can be applied as follows in either a maxRPC or a $1\max\text{RPC}$ algorithm (we use algorithm $(1)\max\text{RPC3}$ for illustration):

1. After a variable x_j is removed from L all neighboring variables x_i are revised. lmaxRPC (resp. maxRPC) will detect a failure if the condition of PC-support loss (resp. either PC-support or PC-witness loss) occurs for all values of x_i . In such situations, the sooner x_i is considered and the failure is detected, the more constraint checks will be saved. Hence, the order in which the neighboring variables of x_j are considered can be determined using a fail-first type of heuristic.
2. Once an AC-support $a_j \in D(x_j)$ has been found for a value $a_i \in D(x_i)$, *search-PCsup* tries to establish if it is a PC-support. If there is no PC-witness for the pair (a_i, a_j) in some variable x_k then a_j is not a PC-support. Therefore, we can again use fail-first heuristics to determine the order in which the variables forming a triangle with x_i and x_j are considered.

The above cases apply to both lmaxRPC and maxRPC algorithms. In addition, a maxRPC algorithm can employ heuristics as follows:

3. For each value $a_i \in D(x_i)$ and each variable x_k constrained with both x_i and x_j , Function 4 checks if the pair (a_i, a_k) still has a PC-witness in $D(x_j)$. If there is no PC-witness or *LastPC* $_{x_i, a_i, x_k}$ is not valid then a new PC-support in x_k is sought. If none is found then a_i will be deleted. Again heuristics can be used to determine the order in which the variables constrained with x_i and x_j are considered.
4. In Function 4 if *LastPC* $_{x_i, a_i, x_k}$ is not valid then a new PC-support for a_i in $D(x_k)$ is sought. The order in which variables constrained with both x_i and x_k are considered can be determined heuristically as in Case 2 above.

As explained, the purpose of such ordering heuristic will be to “fail-first”. That is, to quickly discover potential failures (Case 1 above), refute values that are not PC-supports (Cases 2 and 4) and delete values that have no PC-support (Case 3). Such heuristics can be applied in any coarse-grained maxRPC algorithm to decide the order in which variables are considered in Cases 1-4. Examples are the following:

dom. Consider the variables in ascending domain size. This heuristic can be applied in any of the four cases.

del_ratio. Consider the variables in ascending ratio of the number of remaining values to the initial domain size. This heuristic can be applied in any of the four cases.

wdeg. In Case 1 consider the variables x_i in descending weight for the constraint c_{ij} . In Case 2 consider the variables x_k in descending average weight for the constraints c_{ik} and c_{jk} . Similarly for Cases 3 and 4.

dom/wdeg. Consider the variables in ascending value of dom/wdeg. This heuristic can be applied in any of the four cases.

Experiments demonstrated that applying heuristics in Cases 1 and 3 are particularly effective, while doing so in Cases 2 and 4 saves constraint checks but only marginally reduces cpu times. All of the heuristics mentioned above for Cases 1 and 3 offer cpu gains, with dom/wdeg being the most efficient. Although the primal purpose of the heuristics is to save constraint checks, it is interesting to note that some of the heuristics can also divert search to different areas of the search space when a variable ordering

heuristic like dom/wdeg is used, resulting in fewer node visits. For example, two different orderings of the variables in Case 1 may result in different constraints causing a failure. As dom/wdeg increases the weight of a constraint each time it causes a failure and uses the weights to select the next variable, this may later result in different branching choices. This is explained for the case of AC in [11].

5 Experiments

We have experimented with several classes of structured and random binary CSPs taken from C.Lecoutre’s XCSP repository. Excluding instances that were very hard for all algorithms, our evaluation was done on 200 instances in total from various problem classes. More details about these instances can be found in C.Lecoutre’s homepage. All algorithms used the dom/wdeg heuristic for variable ordering [4] and lexicographic value ordering. In case of a failure (domain wipe-out) the weight of constraint c_{ij} is updated (right before returning in line 15 of Algorithm 1). The suffix ‘+H’ after any algorithm’s name means that we have applied the dom/wdeg heuristic for ordering the propagation list [11], and the same heuristic for *Case 1* described in Section 4. In absence of the suffix, the propagation list was implemented as a FIFO queue and no heuristic from Section 4 was used.

Table 1 compares the performance of stand-alone algorithms used for preprocessing. We give average results for all the instances, grouped into specific problem classes. We include results from the two optimal coarse-grained maxRPC algorithms, maxRPC2 and maxRPC3, from all the light versions of the coarse-grained algorithms, and from one of the most competitive algorithms (maxRPC3) in tandem with the dom/wdeg heuristics of Section 4 (1maxRPC3+H). Results show that in terms of run time our algorithms have similar performance and are superior to existing ones by a factor of two on average. This is due to the elimination of many redundant constraint checks as the cc numbers show. Heuristic do not seem to make any difference.

Tables 2 and 3 compare the performance of search algorithms that apply lmaxRPC throughout search on RLFAPs and an indicative collection of other problems respectively. The algorithms compared are 1maxRPC^{rm} and 1maxRPC3^{rm} with and without

Table 1. Average stand-alone performance in all 200 instances grouped by problem class. Cpu times (t) in secs and constraint checks (cc) are given.

Problem class		maxRPC2	maxRPC3	lmaxRPC2	lmaxRPC3	lmaxRPC ^{rm}	lmaxRPC3 ^{rm}	lmaxRPC3+H
RLFAP (scen_graph)	t	6.786	2.329	4.838	2.043	4.615	2.058	2.148
	cc	31M	9M	21M	8M	21M	9M	8M
Random (modelB_forced)	t	0.092	0.053	0.079	0.054	0.078	0.052	0.056
	cc	0.43M	0.18M	0.43M	0.18M	0.43M	0.18M	0.18M
Geometric	t	0.120	0.71	0.119	0.085	0.120	0.086	0.078
	cc	0.74M	0.35M	0.74M	0.35M	0.74M	0.35M	0.35M
Quasigroup (qcp,qwh,bqwh)	t	0.293	0.188	0.234	0.166	0.224	0.161	0.184
	cc	1.62M	0.59M	1.28M	0.54M	1.26M	0.54M	0.54M
QueensKnights, Queens,QueenAttack	t	87.839	47.091	91.777	45.130	87.304	43.736	43.121
	cc	489M	188M	487M	188M	487M	188M	188M
driver.blackHole haystacks.job-shop	t	0.700	0.326	0.630	0.295	0.638	0.303	0.299
	cc	4.57M	1.07M	4.15M	1.00M	4.15M	1.00M	1.00M

Table 2. Cpu times (t) in secs, nodes (n) and constraint checks (cc) from RLFAP instances. Algorithms that use heuristics are denoted by their name + H. The best cpu time among the lmaxRPC methods is highlighted.

instance		AC^{rm}	$lmaxRPC^{rm}$	$lmaxRPC3^{rm}$	$lmaxRPC^{rm} + H$	$lmaxRPC3^{rm} + H$
scen11	t	5.4	13.2	4.6	12.5	4.3
	n	4,367	1,396	1,396	1,292	1,292
	cc	5M	92M	29M	90M	26M
scen11-f10	t	11.0	29.0	12.3	22.3	9.8
	n	9,597	2,276	2,276	1,983	1,983
	cc	11M	141M	51M	114M	41M
scen2-f25	t	27.1	109.2	43.0	79.6	32.6
	n	43,536	8,310	8,310	6,179	6,179
	cc	44M	427M	151M	315M	113M
scen3-f11	t	7.4	30.8	12.6	17.3	7.8
	n	7,962	2,309	2,309	1,852	1,852
	cc	9M	132M	46M	80M	29M
scen11-f7	t	4,606.5	8,307.5	3,062.8	6,269.0	2,377.6
	n	3,696,154	552,907	552,907	522,061	522,061
	cc	4,287M	35,897M	9,675M	22,899M	6,913M
scen11-f8	t	521.1	2,680.6	878.0	1,902.4	684.7
	n	345,877	112,719	112,719	106,352	106,352
	cc	638M	10,163M	3,172M	7,585M	2,314M
graph8-f10	t	16.4	16.8	9.1	11.0	6.3
	n	18,751	4,887	4,887	3,608	3,608
	cc	14M	71M	31M	51M	21M
graph14-f28	t	31.4	4.1	3.1	2.6	2.1
	n	57,039	2,917	2,917	1,187	1,187
	cc	13M	17M	8M	13M	6M
graph9-f9	t	273.5	206.3	101.5	289.5	146.9
	n	273,766	26,276	26,276	49,627	49,627
	cc	158M	729M	290M	959M	371M

the use of heuristic dom/wdeg for propagation list and for Case 1 of Section 4. We also include results from MAC^{rm} which is considered the most efficient version of MAC [8,9].

Experiments showed that $lmaxRPC^{rm}$ is the most efficient among existing algorithms when applied during search, which confirms the results given in [10]. Accordingly, $lmaxRPC3^{rm}$ is the most efficient among our algorithms. It is between two and four times faster than $maxRPC3^{rm}$ on hard instances, while algorithms $lmaxRPC3$ and $lmaxRPC2$ are not competitive when used during search because of the data structures they maintain. In general, when applied during search, any maxRPC algorithm is clearly inferior to the corresponding light version. The reduction in visited nodes achieved by the former is relatively small and does not compensate for the higher run times of enforcing maxRPC.

Results from Tables 2 and 3 demonstrate that $lmaxRPC3^{rm}$ always outperforms $lmaxRPC^{rm}$, often considerably. This was the case in all 200 instances tried. The use of heuristics improves the performance of both lmaxRPC algorithms in most cases. Looking at the columns for $lmaxRPC^{rm}$ and $lmaxRPC3^{rm}+H$ we can see that our methods can reduce the numbers of constraint checks by as much as one order of magnitude (e.g. in quasigroup problems qcp and qwh). This is mainly due to the elimination of redundant checks inside function *searchPCwit*. Cpu times are not cut down by as much, but a speed-up of more than 3 times can be obtained (e.g. scen2-f25 and scen11-f8).

Importantly, the speed-ups obtained can make a search algorithm that efficiently applies lmaxRPC competitive with MAC on many instances. For instance, in scen11-f10

Table 3. Cpu times (t) in secs, nodes (n) and constraint checks (cc) from various instances

instance		AC^{rm}	$lmaxRPC^{rm}$	$lmaxRPC3^{rm}$	$lmaxRPC^{rm} + H$	$lmaxRPC3^{rm} + H$
rand-2-40-8 -753-100-75	t	4.0	47.3	21.7	37.0	19.0
	n	13,166	8,584	8,584	6,915	6,915
	cc	7M	289M	82M	207M	59M
geo50-20 d4-75-1	t	102.7	347.7	177.5	273.3	150.3
	n	181,560	79,691	79,691	75,339	75,339
	cc	191M	2,045M	880M	1,437M	609M
qcp150-120-5	t	52.1	89.4	50.2	80.0	55.3
	n	233,311	100,781	100,781	84,392	84,392
	cc	27M	329M	53M	224M	36M
qcp150-120-9	t	226.8	410.7	238.1	239.9	164.3
	n	1,195,896	583,627	583,627	315,582	315,582
	cc	123M	1,613M	250M	718M	112M
qwh20-166-1	t	52.6	64.3	38.9	21.2	14.9
	n	144,653	44,934	44,934	13,696	13,696
	cc	19M	210M	23M	53M	6M
qwh20-166-6	t	1,639.0	1,493.5	867.1	1,206.2	816.5
	n	4,651,632	919,861	919,861	617,233	617,233
	cc	633M	5,089M	566M	3,100M	351M
qwh20-166-9	t	41.8	41.1	25.0	39.9	28.5
	n	121,623	32,925	32,925	26,505	26,505
	cc	15M	135M	15M	97M	11M
blackHole 4-4-e-8	t	1.8	14.4	3.8	12.1	3.6
	n	8,661	4,371	4,371	4,325	4,325
	cc	4M	83M	12M	68M	10M
queens-100	t	15.3	365.3	106.7	329.8	103.0
	n	7,608	6,210	6,210	5,030	5,030
	cc	6M	1,454M	377M	1,376M	375M
queenAttacking5	t	34.3	153.1	56.7	136.0	54.8
	n	139,534	38,210	38,210	33,341	33,341
	cc	35M	500M	145M	436M	128M
queensKnights -15-5-mul	t	217.0	302.0	173.6	482.0	283.5
	n	35,445	13,462	13,462	12,560	12,560
	cc	153M	963M	387M	1,795M	869M

Table 4. Average search performance in all 200 instances grouped by class

Problem class		AC^{rm}	$lmaxRPC^{rm}$	$lmaxRPC3^{rm}$	$lmaxRPC^{rm} + H$	$lmaxRPC3^{rm} + H$
RLFAP (scen_graph)	t	242.8	556.7	199.3	416.3	157.3
	cc	233M	2,306M	663M	1,580M	487M
Random (modelB_forced)	t	8.4	28.0	14.8	28.5	17.1
	cc	14M	161M	60M	137M	51M
Geometric	t	21.5	72.2	37.2	57.6	32.1
	cc	39M	418M	179M	297M	126M
Quasigroup (qcp,qwh,bqwh)	t	147.0	162.5	94.9	128.9	89.6
	cc	59M	562M	68M	333M	40M
QueensKnights, Queens,QueenAttack	t	90.2	505.2	180.3	496.4	198.1
	cc	74M	1,865M	570M	1,891M	654M
driver:blackHole haystacks.job-shop	t	3.2	17.1	9.1	11.9	7.0
	cc	1.8M	55M	6.4M	36.7M	5.1M

we achieve the same run time as MAC while $lmaxRPC^{rm}$ is 3 times slower while in scen11-f7 we go from 2 times slower to 2 times faster. In addition, there are several instances where MAC is outperformed (e.g. the graph RLFAPs and most quasigroup problems). Of course, there are still instances where MAC remains considerably faster despite the improvements.

Table 4 summarizes results from the application of $lmaxRPC$ during search. We give average results for all the tested instances, grouped into specific problem classes. As

can be seen, our best method improves on the existing best one considerably, making lmaxRPC outperform MAC on the RFLAP and quasigroup problem classes. Overall, our results demonstrate that the efficient application of a maxRPC approximation throughout search can give an algorithm that is quite competitive with MAC on many binary CSPs. This confirms the conjecture of [6] about the potential of maxRPC as an alternative to AC. In addition, our results, along with ones in [10], show that approximating strong and complex local consistencies can be very beneficial.

6 Conclusion

We presented maxRPC3 and maxRPC3^{rm}, two new algorithms for maxRPC, and their light versions that approximate maxRPC. These algorithms build on and improve existing maxRPC algorithms, achieving the elimination of many redundant constraint checks. We also investigated heuristics that can be used to order certain operations within maxRPC algorithms. Experimental results from various problem classes demonstrate that our best method, lmaxRPC3^{rm}, constantly outperforms existing algorithms, often by large margins. Significantly, the speed-ups obtained allow lmaxRPC3^{rm} to compete with and outperform MAC on many problems. In the future we plan to adapt techniques for using residues from [9] to improve the performance of our algorithms during search. Also, it would be interesting to investigate the applicability of similar methods to efficiently achieve or approximate other local consistencies.

References

1. Balafoutis, T., Stergiou, K.: Exploiting constraint weights for revision ordering in Arc Consistency Algorithms. In: ECAI 2008 Workshop on Modeling and Solving Problems with Constraints (2008)
2. Bessière, C., Régin, J.C., Yap, R., Zhang, Y.: An Optimal Coarse-grained Arc Consistency Algorithm. *Artificial Intelligence* 165(2), 165–185 (2005)
3. Boussemart, F., Hemery, F., Lecoutre, C.: Revision ordering heuristics for the Constraint Satisfaction Problem. In: CP-2004 Workshop on Constraint Propagation (2004)
4. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI-2004 (2004)
5. Debruyne, R., Bessière, C.: From restricted path consistency to max-restricted path consistency. In: Smolka, G. (ed.) CP 1997. LNCS, vol. 1330, pp. 312–326. Springer, Heidelberg (1997)
6. Debruyne, R., Bessière, C.: Domain Filtering Consistencies. *JAIR* 14, 205–230 (2001)
7. Grandoni, F., Italiano, G.: Improved Algorithms for Max-Restricted Path Consistency. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 858–862. Springer, Heidelberg (2003)
8. Lecoutre, C., Hemery, F.: A study of residual supports in arc consistency. In: Proceedings of IJCAI 2007, pp. 125–130 (2007)
9. Likitvivanavong, C., Zhang, Y., Bowen, J., Shannon, S., Freuder, E.: Arc Consistency during Search. In: Proceedings of IJCAI 2007, pp. 137–142 (2007)
10. Vion, J., Debruyne, R.: Light Algorithms for Maintaining Max-RPC During Search. In: Proceedings of SARA 2009 (2009)
11. Wallace, R., Freuder, E.: Ordering heuristics for arc consistency algorithms. In: AI/GI/VI, Vancouver, British Columbia, Canada, pp. 163–169 (1992)

Checking-Up on Branch-and-Check

J. Christopher Beck

Department of Mechanical & Industrial Engineering
University of Toronto
Toronto, Ontario M5S 3G8, Canada
jcb@mie.utoronto.ca

Abstract. Branch-and-Check, introduced ten years ago, is a generalization of logic-based Benders decomposition. The key extension is to solve the Benders sub-problems at each feasible solution of the master problem rather than only at an optimal solution. We perform the first systematic empirical comparison of logic-based Benders decomposition and branch-and-check. On four problem types the results indicate that either Benders or branch-and-check may perform best, depending on the relative difficulty of solving the master problem and the sub-problems. We identify a characteristic of the logic-based Benders decomposition runs, the proportion of run-time spent solving the master problem, that is valuable in predicting the performance of branch-and-check. We also introduce a variation of branch-and-check to address difficult sub-problems. Empirical results show that this variation leads to more robust performance than both logic-based Benders decomposition and branch-and-check on the problems investigated.

1 Introduction

Logic-based Benders decomposition (LBBD) [1,2] has been proposed as a framework for hybrid techniques that combine mixed-integer programming (MIP) and constraint programming (CP). Informally, LBBD requires the decomposition of a problem into a master problem and a set of sub-problems. The solution approach solves the master to optimality, solves the sub-problems, and then adds constraints (“Benders cuts”) to the master problem based on the sub-problem results. The approach iterates between solving the master problem and the sub-problems until a globally optimal solution is found and proved.

Branch-and-Check (B&C) [1,3] is a generalization of LBBD where the sub-problems are solved *during* the search for a solution to the master problem. In Thorsteinsson’s formulation [3], the sub-problems are solved every time a feasible master solution is found, and the cuts are added to the master problem, if necessary. B&C, therefore, is essentially a branch-and-cut search with the Benders sub-problems being the source of the cuts.

Despite the increasing interest in LBBD and speculation (e.g., [4]) that B&C could result in a significant improvement over LBBD, there does not appear to have been a systematic evaluation of B&C. We do not have a clear picture

of how B&C performs on different problems nor what problem characteristics contribute to its behavior. The contributions of this paper, therefore, are:

1. The first systematic empirical comparison of logic-based Benders decomposition and branch-and-check.
2. The identification of a characteristic of the behavior of LBBDD that appears to be correlated to whether an improvement in performance can be expected from B&C.
3. The introduction and evaluation of a variation on B&C that addresses the poor performance of B&C when the sub-problems are difficult.

The following section provides the necessary background. We then turn to the empirical investigations which result in the first two contributions of this paper. Section 4 proposes and evaluates the variation of B&C while Section 5 discusses our results and concludes.

2 Background

In this section, logic-based Benders decomposition and branch-and-check are defined, the literature on branch-and-check is reviewed in detail, and the formal definitions of the problems studied in this paper are presented.

2.1 Logic-Based Benders Decomposition and Branch-and-Check

Logic-based Benders decomposition [2] is a generalization of classical Benders decomposition that is based on the division of a problem into a master problem (MP) and a set of sub-problems (SPs). The MP is a projection of the global model to a subset of decision variables, denoted y , and the constraints and objective function components that involve only y . The rest of the decision variables, x , define the sub-problems. Solving a problem by Benders decomposition involves iteratively solving the MP to optimality and using the solution to fix the y variables, generating the sub-problems. The duals of the SPs are solved to find the tightest bound on the (global) cost function that can be derived from the original constraints and the current MP solution. If this bound is less than or equal to the current MP solution (assuming a minimization problem), then the MP solution and the SP solutions constitute a globally optimal solution. Otherwise, a constraint, called a “Benders cut,” is added to the MP to express the violated bound and another iteration is performed.

Branch-and-check (B&C) [3] moves the SP solving into a branch-and-cut search to solve the MP. Rather than waiting until the MP is solved to optimality, the B&C algorithm solves the SPs at each feasible MP solution, generates the Benders cuts, and immediately adds them to the branch-and-cut tree. The current feasible MP solution is therefore rejected and search continues. Because only globally feasible MP solutions are accepted, the MP is solved only once: there are no MP-SP iterations as in LBBDD.

Intuitively, B&C may out-perform LBBDD because the MP is not repeatedly solved from scratch. Furthermore, a cut introduced based on one sub-optimal MP solution may cut-off other sub-optimal MP solutions whereas in LBBDD these sub-optimal solutions may be enumerated in each iteration.

2.2 Literature Review

Thorsteinsson [3] introduced the name *branch-and-check* and performed experiments on a planning and scheduling problem. A significant speed-up was shown when compared to an LBB model due to Jain & Grossmann [5]. These results were attributed to the claim that the MP was hard to solve, relative to the SPs, and therefore adding the SP cuts based on feasible MP solutions (rather than waiting for optimality) sped-up the overall solving process. A significant weakness of the work, however, is that the B&C implementation did not actually solve the SPs during the MP branch-and-cut search: each time the SPs were solved, the MP search was *restarted*.

Bockmayr & Pizaruk [6] adopt an approach very similar to B&C, except that cuts are added at each node in the MP branch-and-cut tree rather than only at integer feasible nodes [4]. The relaxed solution at each node is used to derive bounds that are then used to define the SPs. Computational results show that solving the SPs more often results in an improvement over the results of Jain & Grossmann. No comparison is done against Thorsteinsson's approach.

Sadykov & Wolsey [7] address the same scheduling problem as the above authors with a B&C approach. They state that solving the SPs at integral MP nodes only is an important feature distinguishing their algorithm from Bockmayr & Pizaruk. However, Sadykov & Wolsey use a tighter MP formulation than Bockmayr & Pizaruk and so it is not possible to attribute their improved performance solely to the more frequent solution of the SPs. In fact, Sadykov & Wolsey state that they believe that the main reason for their performance is the tighter MP model. Follow-up work [8] solves a one-machine minimum weighted number of late activities problem using B&C: the SP is solved at each feasible MP solution and the MP branch-and-cut search continues with the added Benders cuts. However, no LBB algorithm is used for comparison.

In summary, we have been able to find only four papers [3,6,7,8] that have implemented a B&C-like approach. Thorsteinsson and Bockmayr & Pizaruk use the same model as Jain & Grossmann and so these three papers vary only on the frequency with which the sub-problems are solved. Jain & Grossmann solve the sub-problems the least, only when an optimal MP solution is found. Thorsteinsson solves the SP at each feasible MP solution, restarting the MP search each time. Bockmayr & Pizaruk solve the SPs most often, at each node in the MP tree. Both Thorsteinsson and Bockmayr & Pizaruk show better performance than Jain & Grossmann but their approaches have not been compared to each other. The fourth work, Sadykov & Wolsey, also solves the SPs at each feasible MP solution and shows stronger results than Bockmayr & Pizaruk but uses a tighter MP formulation.

It appears, therefore, that the most we can conclude from previous work is that for a specific scheduling problem, solving the SPs more frequently than is done in LBB leads to improved performance. We have not been able to find any work that directly compares B&C (without restarting the MP search) to

¹ The idea of solving the sub-problems *more often* than at each feasible MP solution appears in Hooker's original formulation [1].

LBB. Moreover, work on B&C has been restricted to a single problem type, limiting the generality of the resulting conclusions.

2.3 Problems and Models

In this section, we define the four problems used in this study. Each of these problems has an existing LBB model in the literature. We study three planning and scheduling problems and one location-allocation problem.

CostMinUnary. CostMinUnary is the problem studied by Jain & Grossmann, Thorsteinnsson, and Bockmayr & Pizaruk. The problem is defined by a set of jobs, $j \in J$, each with an individual release date, R_j , and deadline, S_j , which must be scheduled on a set of resources, I . A job can be assigned to any resource; however, its processing time, p_{ij} , and cost, f_{ij} , depend on the resource, $i \in I$, to which it is assigned. The objective is to assign the jobs to resources so that they can execute within their time-windows $[R_j, S_j]$, no jobs on the same resource overlap, and the cost of the resource assignment is minimized.

Following existing work [15,4], the master problem in an LBB model can be defined as follows, with y_{ij} being a 0-1 variable expressing whether job j is assigned to resource i :

$$\text{minimize } \sum_{ij} f_{ij}y_{ij} \tag{1}$$

$$\text{s.t. } \sum_j y_{ij} = 1 \quad \text{all } j \tag{2}$$

$$\sum_j p_{ij}y_{ij} \leq \max_j\{S_j\} - \min_j\{R_j\} \quad \text{all } i \tag{3}$$

$$\sum_{j \in J_{hi}} (1 - y_{ij}) \geq 1 \quad \text{all } i, h = 1, \dots, H - 1 \tag{4}$$

$$y_{ij} \in \{0, 1\} \quad \text{all } i, j$$

The objective function (1) minimizes the cost of assigning jobs to resources, subject to the constraint that all jobs must be assigned to exactly one resource (2). Constraint (3) is a relaxation of the sub-problem expressing that the sum of the durations of the jobs assigned to any resource must be less than or equal to the time between the minimum release date and maximum due date. Constraints (4) are the Benders cuts, where J_{hi} is the set of jobs assigned to resource i in iteration h and that led to an infeasibility in the sub-problem. The cut simply expresses that, in order to form a feasible schedule, at least one job in J_{hi} must be assigned to a different resource.

The sub-problems are then straightforward to define using constraint programming: they are single-machine scheduling problems with release dates and due dates where the goal is to find a feasible schedule. Explicitly, if t_j is the start time of job j , the sub-problem for resource i for all j with $y_{ij} = 1$ is:

$$t_j \geq R_j \tag{5}$$

$$t_j + p_{ij} \leq S_j \tag{6}$$

$$\mathbf{cumulative}(t_j, p_{ij}, \mathbf{1}, \mathbf{1}) \tag{7}$$

The global constraint **cumulative** [9] represents a single-machine scheduling problem to assign values to all start times, t_j , taking into account the durations of each job and the capacities. The capacity required by each job is represented in the vector $\mathbf{1}$ and the capacity of the resource is 1.

CostMinMulti. The CostMinMulti problem is the same as CostMinUnary except that the resources are no longer unary and each job may require more than one unit of the resource. The model [4] has the objective function (1), constraint (2), and the Benders cuts (4) as in CostMinUnary. The sub-problem relaxation (3) is different to account for the discrete resource capacity and the fact that all the problems solved here have a release date of 0 and the same due date, represented by d_0 . Letting C_i be the capacity of resource i and c_{ij} the amount of resource i required by job j during its processing time, the sub-problem relaxation expresses that the area of resource availability (i.e., capacity multiplied by the time horizon: $C_i \times (d_0 - 0)$) must be greater than or equal to the sum of the areas of the jobs assigned to i ($p_{ij}c_{ij}y_{ij}$). Therefore, constraint (8) replaces constraint (3).

$$\frac{1}{C_i} \sum_j p_{ij}c_{ij}y_{ij} \leq d_0, \text{ all } i \tag{8}$$

The sub-problem formulation is also changed to reflect the discrete capacity. Thus, constraint (7) becomes:

$$\mathbf{cumulative}(t_j, p_{ij}, c_{ij}, C_i) \tag{9}$$

MkspMinMulti. MkspMinMulti is a multi-capacity planning and scheduling problem with the objective of makespan minimization. In Hooker’s model [4], M represents the makespan and the master problem is defined as follows:

$$\text{minimize } M \tag{10}$$

$$\text{s.t. } \sum_i y_{ij} = 1 \quad \text{all } j \tag{11}$$

$$M \geq \frac{1}{C_i} \sum_j p_{ij}c_{ij}y_{ij} \quad \text{all } i \tag{12}$$

$$M \geq M_{hi}^* - \sum_{j \in J_{hi}} (1 - y_{ij})p_{ij} \quad \text{all } i, h = 1, \dots, H - 1 \tag{13}$$

$$y_{ij} \in \{0, 1\} \quad \text{all } i, j$$

The differences from CostMinMulti are the sub-problem relaxation (12) and the Benders cut (13). The sub-problem relaxation is, in fact, a restatement of constraint (8) with a variable end of horizon, M , rather than the fixed one, d_0 , and as such is based on exactly the same reasoning. The Benders cut makes

use of M_{hi}^* , the minimum makespan on resource i in iteration h . The expression that is subtracted from M_{hi}^* relies on the fact that the maximum reduction in makespan that can come from removing job j from resource i (i.e., by setting y_{ij} to 0) is the duration of that job, p_{ij} .

Unlike the other two scheduling problems, in MkspMinMulti, the sub-problem is an optimization problem as follows:

$$\text{minimize } M_i \tag{14}$$

$$\text{s.t. } M_i \geq t_j + p_{ij} \tag{15}$$

$$t_j \geq 0 \tag{16}$$

$$\text{cumulative}(t_j, p_{ij}, c_{ij}, C_i) \tag{17}$$

LocAlloc. The LocAlloc problem is a facility location, customer allocation, and truck allocation problem. Given the set J of potential sites and the set I of clients, the goal is to choose which sites to open, to assign each customer to a single open site, to assign a number of trucks to each site, and to assign each customer to a single truck. Multiple customers can be assigned to the same truck provided the sum of their travel distances is less than a given maximum distance for the truck. For each site j there is an opening cost, f_j , and a capacity, b_j . The demand of the clients, d_i , assigned to a site must be less than or equal to the site capacity. Each vehicle has a fixed utilization cost, u , and a maximum total driving distance, ℓ . Serving client i from site j generates a driving distance, t_{ij} , for the vehicle performing the service and has an associated cost, c_{ij} . The available vehicles at a site are indexed in set K with parameter $\bar{k} \geq |K|$ being the maximum number of vehicles at any site.

In the LBB model presented by Fazel-Zarandi & Beck [10], the master problem determines the open sites, the assignment of customers to sites, and the number of trucks at each site. The sub-problems are then separate feasibility problems which attempt to assign the customers to trucks.

The master problem decision variables are:

$$p_j = \begin{cases} 1, & \text{if site } j \text{ is opened} \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if client } i \text{ is served by site } j \\ 0, & \text{otherwise} \end{cases}$$

$numVeh_j$ = number of vehicles assigned to facility j

The master problem can then be modeled as:

$$\text{minimize } \sum_{j \in J} f_j p_j + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + u \sum_{j \in J} numVeh_j \tag{18}$$

$$\text{s.t. } \sum_{j \in J} x_{ij} = 1 \quad i \in I \quad (19)$$

$$\sum_{i \in I} t_{ij} x_{ij} \leq \ell \cdot \bar{k} \quad j \in J \quad (20)$$

$$t_{ij} x_{ij} \leq \ell \quad i \in I, j \in J \quad (21)$$

$$\sum_{i \in I} d_i x_{ij} \leq b_j p_j \quad j \in J \quad (22)$$

$$\text{numVeh}_j \geq \left\lceil \frac{\sum_{i \in I} t_{ij} x_{ij}}{\ell} \right\rceil \quad j \in J \quad (23)$$

$$\text{numVeh}_j \geq \text{numVeh}_{jh}^* - \sum_{i \in I_{jh}} (1 - x_{ij}) \quad j \in J_h \quad (24)$$

$$x_{ij} \leq p_j \quad i \in I, j \in J \quad (25)$$

$$x_{ij}, p_j \in \{0, 1\} \quad i \in I, j \in J \quad (26)$$

The objective (18) is to minimize the total cost of opening facilities, serving customers from a site, and allocating vehicles to a site. Constraint (19) ensures that all clients are served by exactly one facility. The distance limitations are defined by constraints (20) and (21). Constraint (22) limits the demand assigned to facility j . Constraint (23) defines the minimum number of vehicles assigned to each site.

Constraint (24) is the Benders cut. Inspired by the makespan cut in the MkspMinMulti problem, this cut makes use of the optimal number of trucks at facility j in iteration h , numVeh_{jh}^* , and subtracts from it an upper-bound on the reduction in the number of trucks that can arise from removing one customer.

The sub-problem is a feasibility problem to determine if the customers can be feasibly assigned to the allocated trucks. In order to generate a cut, however, we must solve the optimization version of the problem, finding the minimum number of vehicles that the assigned clients can be packed into. This is a bin-packing problem that can be modeled in CP as follows:

$$\min \text{numVehBinPacking}_j$$

$$\text{s.t. } \mathbf{pack}(\text{load}_k, \text{truck}_i, \text{dist}_i) \quad (27)$$

$$\text{numVeh}_j \leq \text{numVehBinPacking}_j < \text{numVehFFD}_j \quad (28)$$

The variables of the sub-problem are load_k , the total travel distance for truck k based on its assigned clients and truck_i , the index of the truck assigned to client i . The distances between site j and client i are represented in the data vector dist_i . The **pack** global constraint (27) maintains the load of the vehicles given the distances and assignments of clients to vehicles [11]. The upper and lower bounds on the number of vehicles are represented by constraint (28). These bounds are derived from the MP solution (numVeh_j) and heuristic preprocessing (numVehFFD_j).

3 A Systematic Evaluation of Branch-and-Check

The next sub-section describes the problem instances for each of our problems as well as providing the experimental details. We then compare logic-based Benders

decomposition and branch-and-check experimentally and present insights into the performance comparison through a deeper analysis of the results.

3.1 Experimental Setup

We use existing problem instances in all of our experiments. For the two multi-capacity scheduling problems we use Hooker’s instances [4]: 75 instances with 2 resources, 60 instances with 3 resources, and 60 instances with 4 resources. The number of jobs varies between 10 and 38. The unary capacity problem instances are generated by modifying the multi-capacity instances by setting the capacity equal to 1 and modifying the time windows of each activity. The overall scheduling horizon is extended by a factor of 3.6, a value chosen after experimentation in order to guarantee that all instances have a feasible solution. The horizon change resulted in two other changes: as in Hooker’s work the possible window for an activity is set to one-third of the (now extended) horizon and, unlike Hooker, the release of each job was drawn with uniform probability from the first two-thirds of the horizon. All other parameters (cost, processing times, etc.) are exactly as in Hooker’s instances.

For the Location-Allocation problems, 300 instances are taken from Fazel-Zarandi & Beck [10]. In half of these instances, the cost of serving a customer from a specific location is correlated with the distance to the location, while in the remaining half, distance and cost are uncorrelated. The problem sizes (i.e., number of possible facilities \times number of clients) are: $\{20 \times 10, 30 \times 15, 40 \times 20\}$.

All experiments were run with a 7200-second time limit on a Duo Core AMD 720 CPU with 1 MB cache, 4 GB of memory, running Red Hat Enterprise 4. The MIP solver is CPLEX 12.1 and the CP solver is ILOG Solver/Scheduler 6.7.

3.2 Logic-Based Benders Decomposition vs. Branch-and-Check

The comparison of logic-based Benders decomposition with branch-and-check is shown in Table 12. For each problem set, we present the mean and median difference in CPU time (LBBBD minus B&C). This formulation means that positive numbers favor B&C (i.e., it has a lower mean CPU time) and negative entries favor LBBBD. Using a bootstrap paired- t test [12], we also indicate the statistical significance at $p \leq 0.005$.

Our results are consistent with previous work on the CostMinUnary problems: B&C shows a clear benefit, especially with an increased number of resources. However, the advantage for B&C disappears for the other scheduling problems to the point that LBBBD shows significantly lower mean run-time overall and on three of the six subsets of CostMinMulti and MkspMinMulti. Finally, for LocAlloc, B&C again shows a significant advantage over LBBBD.

3.3 A Deeper Analysis

Table 2 presents further data: the number of iterations and the percentage of the run-time spent on the master problem and the sub-problems.

² The OPT15 columns are discussed in Section 4.1.

Table 1. Summary of B&C and OPT15 Performance. Mean and Median are the corresponding average differences in run-time (in seconds) between B&C and LBB and between OPT15 and LBB. A negative value indicates that LBB achieves a lower run-time. The symbols * and † indicate a significant difference in mean run-time at $p \leq 0.005$, for the corresponding B&C variation and LBB, respectively.

Problem	Set	B&C		OPT15	
		Mean	Median	Mean	Median
CostMinUnary	2	*110.9	0.1	*110.9	0.1
	3	*164.8	1.2	*200.2	1.2
	4	*1049.7	19.7	*1038.7	19.7
	all	*416.3	0.7	*423.8	0.7
CostMinMulti	2	†-206.4	0	†-207.3	0
	3	-194.8	0.1	†-224.6	0.1
	4	-15.1	0.9	-8.6	0.9
	all	†-144.0	0	†-151.5	0
MkspMinMulti	2	-106.8	0	-58.7	0
	3	†-361.7	0	-215.2	0
	4	†-804.3	-0.1	-163.9	0.2
	all	†-400.0	0	-139.2	0
LocAlloc	cor	*999.4	66.8	*948.1	12.9
	uncor	*812.5	11.4	*848.5	11.7
	all	*905.9	23.9	*898.3	12.4

Table 2. Details of the logic-based Benders decomposition vs. Branch-and-Check experiment. On the left-hand side, we present the mean (and median) number of master problem iterations and the percentage of time spent solving the master problem (% MP) and the sub-problems (% SP). On the right-hand side, branch-and-check data is presented: the number of iterations of the sub-problem (i.e., the number of times that the *set* of sub-problems is solved—recall that the master problem is solved only once), as well as the percentages of the run-time spent on the master problem and sub-problems.

Problem	Set	LBB			B&C		
		Iterations	% MP	% SP	SP Iterations	% MP	% SP
CostMinUnary	2	62.8 (6)	75.9 (91.5)	22.8 (7.9)	165.4 (11)	32.0 (28.0)	57.3 (64.5)
	3	138.9 (24)	90.4 (95.1)	9.6 (4.9)	1047.5 (40)	44.1 (42.7)	54.2 (57.1)
	all	146.3 (23)	86.6 (97.0)	12.9 (2.8)	700.5 (50)	42.4 (42.5)	53.0 (55.6)
CostMinMulti	2	2.6 (1)	12.8 (0)	81.9 (100)	4.1 (2)	8.1 (0)	90.5 (100)
	3	23.6 (14.5)	51.3 (66.5)	48.7 (33.5)	47.2 (26.5)	15.9 (3.7)	84.1 (96.2)
	4	35.2 (22)	68.8 (93.4)	31.2 (6.6)	69.4 (45.5)	22.7 (18.1)	77.3 (81.9)
	all	19.1 (7)	41.9 (11.2)	56.1 (73.5)	37.4 (17)	15.0 (0.5)	84.5 (99.4)
MkspMinMulti	2	18.9 (5)	9.0 (0)	91.0 (100)	20.8 (6)	3.7 (0)	96.3 (100)
	3	59.0 (25.5)	37.4 (33.3)	62.6 (66.7)	69.9 (28.5)	13.0 (1.2)	87.0 (98.8)
	4	51.3 (20)	55.5 (60.0)	44.5 (40.0)	70.2 (36)	17.4 (8.7)	82.6 (91.4)
	all	41.2 (13)	32.0 (21.5)	68.0 (78.5)	51.1 (20)	10.8 (0.3)	89.2 (99.7)
LocAlloc	cor	7.2 (1.5)	99.9 (100)	0.1 (0)	70.6 (19)	99.4 (100)	0.6 (0)
	uncor	5.9 (2)	99.9 (100)	0.1 (0)	87.2 (23.5)	98.4 (100)	1.6 (0)
	all	6.5 (2)	99.9 (100)	0.1 (0)	78.9 (21.5)	98.9 (100)	1.1 (0)

The statistics for LBBD indicate that it has significantly different behavior on the four problems. Using the median, we see that in CostMinUnary, LBBD spends over 97% of the run-time solving the MP and does 23 iterations. This is a substantial difference from the other two scheduling problems: a median 11% and 21% of their run-times is spent on the MP and they perform a median of 7 and 13 iterations, respectively. LocAlloc is different again, spending 100% of the run-time on the master problem but only requiring two iterations.

The branch-and-check results show a large increase in both the number of times that the SPs are solved and a corresponding increase in the proportion of CPU time spent solving them. This pattern is not seen for LocAlloc as, though there is a substantial increase in the number of SP iterations, most of the run-time is still spent solving the master problem.

The positive differences in CPU time in Table 1 correspond to problem sets where a significant portion of the run-time is spent on the master problems. Figure 1 plots the mean difference in run-time between LBBD and B&C against the proportion of time spent solving the MP by LBBD. We have aggregated the latter data into 10 buckets corresponding to intervals of size 0.1. The pattern that can be observed is that unless LBBD spends about 80% or more of its time solving the master problem, the benefits from branch-and-check are rare. In contrast, with master run-time proportions approaching 1, both the magnitude and the frequency of benefits from using B&C are much higher.

These results can be understood by noting that LBBD and B&C embody different expectations with respect to relative sub-problem difficulty. In LBBD, the SPs are solved once for every (optimal) MP solution. In B&C, the SPs are solved at each feasible solution to the MP. If the MP is much harder to solve than the SPs, solving the MP once and using the cuts that are generated inexpensively from repeated SP solutions should result in lower overall run-time. In contrast, if the SPs are not easily solved, then frequently solving them is counter-productive. It would be better to solve the SPs only when necessary: when an optimal master problem needs to be either confirmed or cut-off. This is precisely the link between the results in Tables 1 and 2.

The generality and analytical understanding of this pattern remain to be explored. However, we believe that, as a broad measure, the portion of run-time spent by LBBD on solving the master problem is a promising indicator of the benefit that can accrue from applying B&C. Minimally, it can be employed by practitioners when they are deciding if spending the time to implement B&C is likely to be worthwhile.

4 A Variation on Branch-and-Check

The experiments above indicate that the difficulty in solving sub-problems is important to the performance differences between LBBD and B&C. Additionally, we make two observations.

1. A feasible MP solution may be very different from an optimal one. The cuts that are generated to remove the former may be irrelevant to cutting off

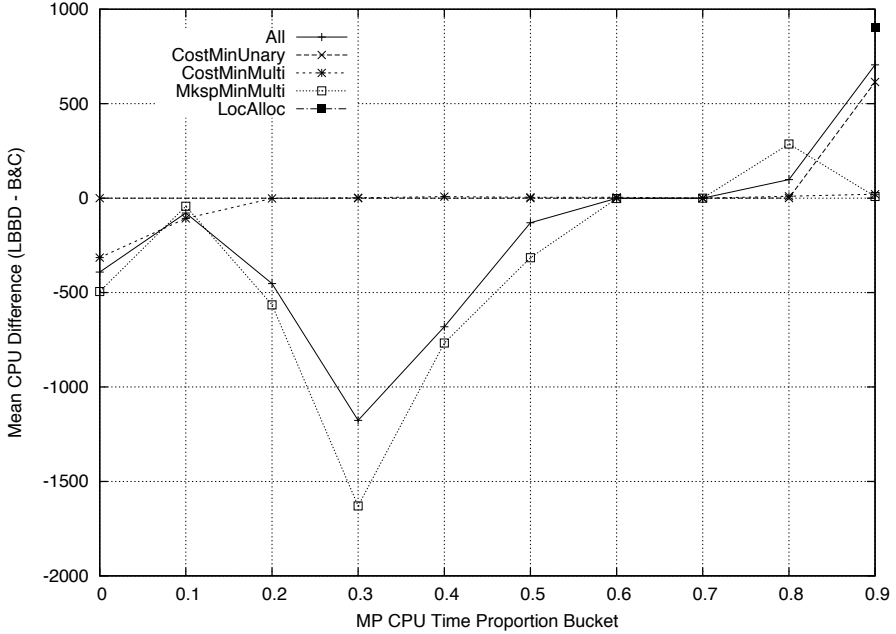


Fig. 1. A plot of the proportion of run-time spent solving the master problem, aggregated into 10 buckets ($[0, 0.1)$, $[0.1, 0.2)$... $[0.9, 1.0]$), against the mean difference in CPU time (LBB minus B&C). Note that the LocAlloc data all fall into the final bucket and therefore form only a single point.

optimal MP solutions that are not globally feasible and, therefore, the work of solving the SPs and generating cuts may be wasted.

2. The sub-problems in a given problem instance are not equally difficult. For example, on the scheduling problems we have observed poor but feasible MP solutions that place most or all activities on one machine, inducing the worst-case in terms of SP difficulty.

These observations suggest that the avoidance of difficult and irrelevant sub-problems may lead to better performance. Therefore, we propose a variation of B&C that solves the SPs more frequently than LBB but less often than B&C by filtering the feasible MP solutions for which it solves the SPs. Our simple idea, denoted OPT15, is as follows: within the B&C algorithm, rather than solving SPs for each feasible MP solution, we solve the SPs corresponding to feasible MP solutions with an optimality gap of less than 15%³. Feasible solutions with larger gaps are accepted as globally feasible.

The completeness of B&C is compromised by this change unless a feasible MP solution with a gap of less than 15% is subsequently found *and* proved

³ This gap is between the cost of the incumbent MP solution and the best current lower bound on the MP solution.

Table 3. The mean (and median) number of sub-problem iterations (SP Iter.) and percentage of problem instances (% Best) for which the algorithm’s run-time was within 10 seconds of the best run-time. Bold entries indicate the highest percentage in a row.

Problem	Set	LBBD		B&C		OPT15	
		% Best	SP Iter.	% Best	SP Iter.	% Best	SP Iter.
CostMinUnary	2	85.3	62.8 (6)	100.0	165.4 (11)	100.0	165.0 (11)
	3	75.0	138.9 (24)	98.3	1047.5 (40)	98.3	1008.3 (34)
	4	41.7	258.2 (81.5)	96.7	1022.4 (160)	91.7	1045.7 (139)
	all	68.7	146.3 (23)	98.4	700.5 (50)	96.9	695.4 (40)
CostMinMulti	2	98.7	2.6 (1)	88.0	4.1 (2)	88.0	4.1 (2)
	3	83.3	23.6 (14.5)	76.7	47.2 (26.5)	78.3	45.9 (26.5)
	4	80.0	35.2 (22)	81.7	69.4 (45.5)	85.0	74.5 (41.5)
	all	88.2	19.1 (7)	82.6	37.4 (17)	84.1	38.6 (16)
MkspMinMulti	2	90.7	18.9 (5)	85.3	20.8 (6)	86.7	19.5 (6)
	3	85.0	59.0 (25.5)	68.3	69.9 (28.5)	80.0	62.6 (26.5)
	4	88.3	51.3 (20)	66.7	70.2 (36)	85.0	71.6 (26)
	all	88.2	41.2 (13)	74.4	51.1 (20)	84.1	48.8 (17)
LocAlloc	cor	35.3	7.2 (1.5)	82.0	70.6 (19)	71.3	62.7 (15.5)
	uncor	45.3	5.9 (2)	74.0	87.2 (23.5)	84.0	69.3 (20)
	all	40.3	6.5 (2)	78.0	78.9 (21.5)	77.7	66.0 (18)
All		67.7	47.8 (6)	82.7	200.9 (23)	84.7	194.9 (19)

to be globally feasible. If such a new MP solution is not found, we preserve completeness by running a second iteration of B&C without the 15% threshold. The second iteration has two significant advantages over the first iteration: all the cuts from solving the SPs in the first iteration are incorporated and the warm-start functionality of the MIP solver typically allows a good initial feasible MP solution to be found in the pre-solve phase.

The choice of 15% is arbitrary and based on examination of preliminary experiments. No tuning was done to investigate different choices for the threshold.

4.1 Experimental Evaluation

The right-hand side of Table 1 in Section 3.2 presents the mean and median run-time difference between LBBD and OPT15. The problem instances and experimental setup are the same as described in Section 3.1.

The empirical results indicate that OPT15 performs more robustly than LBBD or B&C. On the problems where B&C does significantly better than LBBD (CostMinUnary and LocAlloc), OPT15 performs about the same as B&C, achieving a statistically significant difference when compared to LBBD run-time and achieving equivalent performance in terms of mean run-time as B&C. The only statistic that is significantly different is the median run-time for LocAlloc, which is considerably smaller for OPT15. On problems where B&C performs poorly compared to LBBD, OPT15 performs approximately the same as B&C on CostMinMulti and much better on MkspMinMulti. In fact, there is no statistically significant difference between LBBD and OPT15 on the latter set.

A different perspective on this data is presented in Table 3. The table presents two pieces of data: the mean and median number of times that the set of SPs is solved (SP Iter.) and the percentage of problem instances for which the algorithm achieved a run-time within 10 seconds of the best run-time achieved by any algorithm. For example, on the two-machine instances of CostMinUnary, LBBB is within the 10 seconds of the best run-time on 85.3% of the problem instances, while B&C and OPT15 are within the threshold on all instances.

The SP iteration data demonstrate that, indeed, OPT15 tends to solve the sub-problems less frequently than B&C. The difference, however, is small.

The % Best data indicates that OPT15 is seldom best on a given subset: it is alone with the highest percentage on two subsets (CostMinMulti/4 and LocAlloc/uncor), while LBBB is uniquely the best on 5 sets and B&C on 2. However, it is never the worst performer while LBBB and B&C have poor results on different problem sets. Overall, the performance of OPT15 results in it being within the 10-second threshold on 84.7% the problems compared to 82.7% and 67.7% for B&C and LBBB, respectively.

5 Discussion and Conclusion

This paper has presented the first systematic comparison of logic-based Benders decomposition and branch-and-check. Using four different problems from the scheduling and facility location literature, we have demonstrated that B&C can lead to a significant improvement over LBBB but that the improvement is dependent on the difficulty of solving the sub-problems relative to that of solving the master problem. For problems where the sub-problem is difficult, B&C can result in significantly longer run-times than LBBB. We have also shown that the proportion of run-time used in LBBB to solve the master problem is a good measure of the likelihood of the benefit from implementing a B&C algorithm. Our results show that unless at least 80% of the LBBB run-time is spent on the master problem, benefits from B&C are small and rare.

The generality of these conclusions is still in question, as we have only evaluated four problem types, three of which are related scheduling problems. It would be interesting to perform similar experiments with radically different problems. Despite the problem similarities, however, three different behaviors were observed in terms of the proportion of CPU time spent solving the sub-problems and the number of master problem iterations (see Table 2). Furthermore, our results are consistent with our understanding of the increased emphasis on solving sub-problems that is embodied by B&C. We are, therefore, optimistic that the conclusions here will be confirmed in follow-up research.

In our experiments, the sub-problem relaxation and the Benders cut were not independent variables. These are two critical components of an LBBB-style algorithm and changing these model components may change the relative performance of LBBB and B&C on a given problem. However, we conjecture that the fundamental conclusion regarding the proportion of effort in solving the master problem versus the sub-problems would still be valid. Specifically, a tighter,

harder-to-compute Benders cut should result in fewer iterations but may result in much more expensive sub-problems. Depending on the strength and computational cost of the cut, the new model may spend either a higher or lower proportion of its run-time on the sub-problems. Our results suggest that if the new cut shifts the run-time toward the master problem then B&C has an improved likelihood of out-performing LBBDD when compared with the weaker cut. The reverse is true if the new cut results in a larger proportional effort on the sub-problems. In contrast, a tighter, more expensive sub-problem relaxation should increase the effort required in solving the master problem while reducing the number of times that the sub-problems must be solved. This shift suggests that a tighter sub-problem relaxation would tend to favour B&C over LBBDD.

This paper also introduces OPT15, a B&C variation that achieves more robust performance by avoiding sub-problems that are difficult and irrelevant to cutting off optimal master solutions. It achieves this goal by only solving sub-problems for master problem solutions with an optimality gap of 15% or less.

The relative performance of OPT15 and B&C depends on the quality of the feasible MP solutions that are found. Experiments using an earlier version of CPLEX (version 11.0) demonstrated significantly worse B&C performance and correspondingly larger OPT15 improvement on the CostMinMulti and Mksp-MinMulti problems. The performance change with CPLEX 12.1 was due to an improvement in the quality of the first feasible MP solution found. With CPLEX 11.0, the initial MP solutions often induced worst-case SPs that, by themselves, exhausted the 7200-second time limit. It would be interesting to repeat the above experiments with different CPLEX settings (e.g., preferring optimal to feasible solutions) and with other MIP solvers to further investigate the importance of the initial feasible MP solution. We expect the performance of OPT15 to increase when the initial feasible MP solutions are of poorer quality.

OPT15 investigates “middle ground” between solving sub-problems only for optimal MP solutions versus solving them for each feasible MP solution. As a relatively simple idea, it is unclear if OPT15 specifically deserves further development. However, as an example of a technique that interpolates between LBBDD and B&C, it opens the possibility for more sophisticated approaches. Our choice of using a threshold on the optimality gap and the specific choice of that threshold were arbitrary. One might instead set a small time-limit on sub-problem searches. For easy sub-problems, the performance would be identical to B&C while for harder sub-problems, performance may approach that of LBBDD. One could consider adaptively learning such time limits for given problems or problem instances.

Acknowledgments. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, Canadian Foundation for Innovation, Ontario Ministry for Research and Innovation, Alcatel-Lucent, and IBM. Thanks to Daria Terekhov for comments on earlier versions.

References

1. Hooker, J.N.: *Logic-based Methods for Optimization*. Wiley, Chichester (2000)
2. Hooker, J., Ottosson, G.: Logic-based Benders decomposition. *Mathematical Programming* 96, 33–60 (2003)
3. Thorsteinsson, E.S.: Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 16–30. Springer, Heidelberg (2001)
4. Hooker, J.: A hybrid method for planning and scheduling. *Constraints* 10, 385–401 (2005)
5. Jain, V., Grossmann, I.E.: Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing* 13(4), 258–276 (2001)
6. Bockmayr, A., Pizaruk, N.: Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. *Computers & Operations Research* 33, 2777–2786 (2006)
7. Sadykov, R., Wolsey, L.A.: Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. *INFORMS Journal on Computing* 18(2), 209–217 (2006)
8. Sadykov, R.: A branch-and-check algorithm for minimizing the weighted number of late jobs on a single machine with release dates. *European Journal of Operational Research* 189, 1284–1304 (2008)
9. Baptiste, P., Le Pape, C., Nuijten, W.: *Constraint-based Scheduling*. Kluwer Academic Publishers, Dordrecht (2001)
10. Fazel-Zarandi, M.M., Beck, J.C.: Solving a location-allocation problem with logic-based Benders decomposition. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 344–351. Springer, Heidelberg (2009)
11. Shaw, P.: A constraint for bin packing. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 648–662. Springer, Heidelberg (2004)
12. Cohen, P.R.: *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge (1995)

Spatial, Temporal, and Hybrid Decompositions for Large-Scale Vehicle Routing with Time Windows

Russell Bent¹ and Pascal Van Hentenryck²

¹ Los Alamos National Laboratory

² Brown University

Abstract. This paper studies the use of decomposition techniques to quickly find high-quality solutions to large-scale vehicle routing problems with time windows. It considers an adaptive decomposition scheme which iteratively decouples a routing problem based on the current solution. Earlier work considered vehicle-based decompositions that partitions the vehicles across the subproblems. The subproblems can then be optimized independently and merged easily. This paper argues that vehicle-based decompositions, although very effective on various problem classes also have limitations. In particular, they do not accommodate temporal decompositions and may produce spatial decompositions that are not focused enough. This paper then proposes customer-based decompositions which generalize vehicle-based decouplings and allows for focused spatial and temporal decompositions. Experimental results on class R2 of the extended Solomon benchmarks demonstrates the benefits of the customer-based adaptive decomposition scheme and its spatial, temporal, and hybrid instantiations. In particular, they show that customer-based decompositions bring significant benefits over large neighborhood search in contrast to vehicle-based decompositions.

1 Introduction

The scale of optimization problems and the need for finding high-quality solutions has grown steadily in recent years as optimization systems are increasingly deployed in operational, integrated settings. This trend generates significant issues for optimization research, changing the focus from finding optimal solutions to delivering high-quality solutions under time constraints. This paper examines the underlying algorithmic issues in the context of multiple vehicle routing with time windows (VRPTWs), which arise in many transportation applications including courier services, the scheduling of repairs in telecommunication companies, and supply-chain logistics. VRPTWs are particularly interesting in this respect, since instances with as few as 100 customers have not been solved optimally despite intense research. Hence finding high-quality solutions under time constraints for problems with 1,000 customers is a significant challenge.

Spatial and temporal decouplings [17] are natural avenues for speeding up optimization algorithms. Unfortunately, they do not apply easily to large-scale VRPTWs that involve complex spatial and temporal dependencies. To remedy this limitation, the concept of adaptive decoupling was proposed in [4]. Its key idea is to iteratively select subproblems that are optimized independently and reinserted into an existing solution. The successive decouplings are adaptive as they depend on the current solution, not simply

the instance data. The benefits of this approach were demonstrated by a vehicle-based adaptive spatial decomposition VASD scheme which produces high-quality solutions significantly faster than large neighborhood search (LNS) on the class RC1 of the extended Solomon benchmarks. Informally speaking, the VASD scheme partitions the vehicles of an existing solution to obtain two subproblems, reoptimizes one of these subproblems using LNS, and reinsert the optimized vehicle routes to obtain a new solution. The VASD scheme is attractive since it makes it easy to merge the solutions of decoupled problems. However, it also has a number of limitations. Because it is vehicle-based, it is not as spatially focused as possible since vehicles may often travel across large regions, especially early in the optimization process. Moreover, vehicle-based decompositions cannot really accommodate temporal decouplings, since vehicles generally serve customers with a wide variety of time windows.

This paper remedies these limitations and proposes a customer-based adaptive decomposition (CAD) scheme which can be naturally instantiated to spatial, temporal, and hybrid decouplings. Its key idea is to select a set of customers based on a spatial, temporal, or hybrid property and to define a generalized multi-depot VRPTW involving these customers only. The CAD scheme thus allows for more focused spatial decompositions, tight temporal decompositions, or a combination thereof. The generalized VRPTW is also designed to allow for an easy merging of its reoptimized solution into the existing solution.

The benefits of the CAD scheme are demonstrated on the class R2 of the extended Solomon benchmarks. The experimental results indicate that the CAD scheme significantly outperforms LNS and the VASD scheme on this class. They also indicate the complementarity between spatial and temporal decompositions and hence the value of hybrid decompositions.

The rest of this paper is organized as follows. It first reviews VRPTWs and the adaptive decomposition scheme. It then presents the earlier work on vehicle-based adaptive spatial decompositions and the novel contributions on customer-based adaptive decouplings. The paper then presents several instantiations of the CAD scheme, including spatial, temporal, and randomized decouplings. The experimental results and the related work concludes the paper.

2 VRPTWs

A VRPTW instance is specified by a set \mathcal{C} of customers, a set of departure depots \mathcal{D}^- , a set of arrival depots \mathcal{D}^+ , and a set of vehicles \mathcal{V} such that $|\mathcal{D}^-| = |\mathcal{D}^+| = |\mathcal{V}|$. A single depot problem is easily generalized into a multi-depot problem by creating multiple depots at the same location. We use multiple depots since it enables us to specify decoupled problems as VRPTWs. The sites of the VRPTW instance are elements of $Sites = \mathcal{C} \cup \mathcal{D}^- \cup \mathcal{D}^+$. Every site c has a demand $q_c \geq 0$ and a service time $s_c \geq 0$ which is the duration spent at each customer location. The travel cost between sites i and j is t_{ij} . Each site c has a time window $[e_c, l_c]$ constraining when it can be visited, where e_c and l_c represent the earliest and latest arrival times. Vehicles must arrive at site c before the end of the time window l_c . They may arrive early but they have to wait until time e_c to be serviced. Each vehicle has a capacity Q . The recursive computation

of earliest and latest arrival times is specified in detail in [2] and is omitted here for space reasons.

Solutions are specified in terms of vehicle routes and routing plans. A vehicle route starts from a depot d^- , visits a number of customers at most once, and returns to a depot d^+ . It is thus a sequence $\langle d^-, c_1, \dots, c_n, d^+ \rangle$ where all sites are different. The customers of a route $r = \langle d^-, c_1, \dots, c_n, d^+ \rangle$, denoted by $cust(r)$, is the set $\{c_1, \dots, c_n\}$ and the route r of a customer in $\{c_1, \dots, c_n\}$ is denoted by $route(c)$. The size of a route, denoted by $|r|$, is $|cust(r)|$. The demand of a route, denoted by $q(r)$, is the sum of the demands of its sites, i.e., $q(r) = q(d^-) + \sum_{i=1}^n q(c_i) + q(d^+)$. A route satisfies its capacity constraint if $q(r) \leq Q$. We use $q(c)$ to denote the amount of capacity used by a route up to site c . The travel cost $t(r)$ of a route $r = \langle d^-, c_1, \dots, c_n, d^+ \rangle$ is the cost of visiting all its sites, i.e., $t(r) = d(d^-, c_1) + d(c_1, c_2) + \dots + d(c_{n-1}, c_n) + d(c_n, d^+)$.

A routing plan is a set of routes in which every customer is visited exactly once and every depot at most once. Observe that a routing plan assigns a unique earliest arrival time a_c for each site c . It also assigns a unique return time $a(r)$ to its destination depot d^+ for each route r . The routing plan also assigns a departure time for each site c , denoted by δ_c . The routing plan also assigns a critical arrival time for each site c , denoted by z_c . This is the latest time a vehicle can feasibly arrive at c .

A solution to the VRPTW is a routing plan σ satisfying the capacity and time window constraints, i.e., $\forall r \in \sigma : q(r) \leq Q$ & $\forall c \in Sites : a_c \leq l_c$. The ordering of the customers on a route in σ implicitly defines a predecessor and successor site for each site c , denoted by $pred(\sigma, c)$ and $succ(\sigma, c)$. When the context is clear, σ is dropped from the notation for brevity. The size $|\sigma|$ of a routing plan σ is the number of non-empty routes in σ . The VRPTW problem consists of finding a solution σ which minimizes a lexicographic function consisting of the number of vehicles and the total travel cost, i.e., $f(\sigma) = \langle |\sigma|, \sum_{r \in \sigma} t(r) \rangle$. Modern algorithms for the VRPTW are often organized in two stages, first minimizing the number of vehicles and then minimizing travel distance [2][9].

3 The Adaptive Decomposition Scheme

This paper aims at finding decouplings to speed up the solving of large-scale VRPTWs. The goal of the decouplings is to decompose a VRPTW \mathcal{P} into two sub-VRPTWs \mathcal{P}_o and \mathcal{P}_s that can be solved independently and whose solutions can be merged into a solution of \mathcal{P} . In general, finding static decompositions is difficult. For this reason, we proposed in [4] to use the current solution σ of \mathcal{P} to find a decoupling $(\mathcal{P}_o, \mathcal{P}_s)$ with projected solution σ_o and σ_s . The VRPTW \mathcal{P}_o is then reoptimized and its solution is merged with σ_s to obtain a new solution to \mathcal{P} . More precisely, the Adaptive Decomposition Scheme (ADS) is based on two main principles:

1. Starting from plan σ_0 , it produces a sequence of plans $\sigma_1, \dots, \sigma_j$ such that $f(\sigma_0) \geq f(\sigma_1) \geq \dots \geq f(\sigma_j)$.
2. At step i , the scheme uses σ_{i-1} to obtain a decoupling $(\mathcal{P}_o, \mathcal{P}_s)$ of \mathcal{P} with projected solutions σ_o and σ_s . It reoptimizes \mathcal{P}_o to obtain σ_o^* and the new plan $\sigma_i = \text{MERGE}(\sigma_o^*, \sigma_{i-1})$

One of the most challenging aspects of ADS is how to perform the merging of the decoupled solutions, i.e., $\sigma_i = \text{MERGE}(\sigma_o^*, \sigma_{i-1})$. In [4], we addressed this challenge by choosing \mathcal{P}_o such that the customers of entire vehicles are removed. The merging operation is then trivial, since the vehicles in $(\mathcal{P}_o$ and $\mathcal{P}_s)$ are disjoint. We now review this scheme to emphasize its strengths and limitations.

4 Vehicle-Based Spatial Adaptive Decompositions

The decomposition presented in [4] is a vehicle-based adaptive decoupling (VAD). It partitions the vehicles to obtain \mathcal{P}_o and \mathcal{P}_s , reoptimizes \mathcal{P}_o , and uses the new optimized routes, and the routes in \mathcal{P}_s to obtain a new solution. Only spatial decompositions were considered in [4]. The idea was to view the customer region as a circle, randomly selects a wedge W , and partitions the vehicles into those serving at least one customers in W and the others. The resulting Vehicle-Based Spatial Adaptive Decomposition VASD is particularly effective and produced high-quality solutions quickly on instances with up to 1,000 vertices. Its main benefits are the simple definition of \mathcal{P}_o and the trivial implementation of merging, which simply uses the optimized routes of \mathcal{P}_o to replace the old routes in the existing solution.

The VAD scheme has a number of limitations however. First, because the decoupling is vehicle-based, the customers can be located significantly outside the selected wedge. This is illustrated in Figure 1 which depicts the behavior of the VASD scheme visually. The left part of Figure 1 shows the initial plan σ_0 (left) and the plan σ_1 (right) after the first decoupling and optimization. The customers in the subproblem \mathcal{P}_o are in red, the remaining ones in blue. The right part of Figure 1 shows the projected solution σ_o for subproblem \mathcal{P}_o (left) and its reoptimization σ_o^* (right). As can be seen, the first subproblem is quite spread out, illustrating the spatial decomposition is not as tight as desired.

More important however is the fact that the VAD scheme does not scale to other decomposition criteria and, in particular, to temporal decompositions. Indeed, unless the time windows are wide, it is very unlikely that good solutions cluster customers with similar time windows on the same vehicle, since the vehicle will be inactive for most

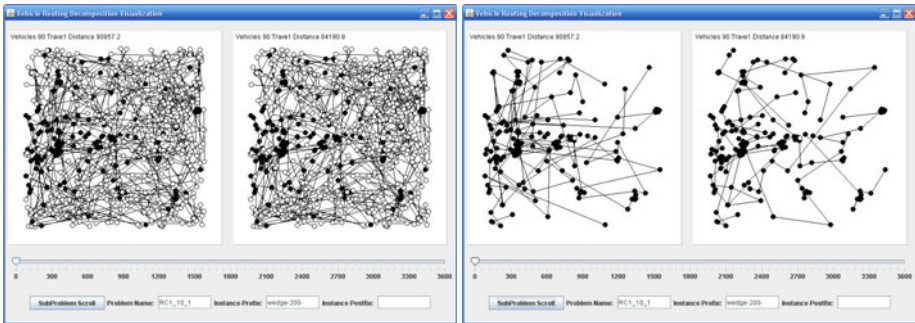


Fig. 1. The First Decoupling of VASD

of the time horizon. Since it uses a vehicle-based decomposition, the VASD scheme is not well-adapted to exploit temporal locality.

5 Customer-Based Adaptive Decompositions

To remedy this limitation, this paper proposes a Customer-based Adaptive Decomposition (CAD) scheme. A decoupled problem in the CAD scheme is given by a set of customer sequences and has a new set of depots and constraints so that the solutions of σ_o^* can be inserted into σ_s , while ensuring feasibility of the resulting plan.

Given a sequence of customers $\langle c_i, \dots, c_j \rangle$ for the decoupling, the depots of the subproblem are constructed as follows:

- $d^- = \text{pred}(c_i)$: the origin depot is the predecessor of the sequence.
- $d^+ = \text{succ}(c_j)$: the destination depot is the successor of the sequence.
- $e_{d^-} = \delta_{\text{pred}(c_i)}$: the departure time of c_i is the earliest departure time for d^- .
- $l_{d^+} = z_{\text{succ}(c_j)}$: the critical arrival time of $\text{succ}(c_j)$ is the latest arrival for d^+ .
- $q_{d^-} = q(\text{pred}(c_i))$: the demand of d^- is the cumulative demand up to $\text{pred}(c_i)$.
- $q_{d^+} = q(\text{succ}(c_j)) - q(c_j)$: the demand of d^+ is the cumulative demand after c_j .

By constructing depots using the border regions of a sequence, any feasible route between d^- and d^+ can be reinserted between $\text{pred}(c_i)$ and $\text{succ}(c_j)$ of σ_{i-1} , while maintaining the feasibility of \mathcal{P}_i .

The CAD scheme is formalized in Figure 2. The core of the algorithm is in lines 3–6 which selects a set of customers (line 3), extracts the customers as a VRPTW (line 4), reoptimizes subproblem \mathcal{P}_o using algorithm \mathcal{A} (line 5), and merges the new optimized subplan σ_o^* to obtain the new solution (line 6). These main steps are repeated until the time limit is reached. The extraction step is given by the EXTRACT function, which collects all vehicles serving a customer in the decomposition (line 1), collects all the customers served by these vehicles in between customers of S , and constructs the depots (lines 2–10). The customers and depots so obtained define the subproblem (line 11). The CONSTRUCTARRIVALDEPOT and CONSTRUCTDEPARTUREDEPOT functions describe how to create depots for \mathcal{P}_o that allows σ_o^* to be feasibly merged into σ . Finally, the MERGE function shows how σ_o^* is merged into σ .

6 Instantiations of the CAD Scheme

This section presents a variety of instantiations of the CAD scheme. Each such instantiation only has to specify how the function SELECTCUSTOMERS is implemented. We start with the vehicle-based spatial decomposition proposed in [4], generalize it, and then present temporal and random decompositions.

6.1 The VASD Scheme

We first show how the VASD scheme can be viewed as an instantiation of CAD. The VASD decomposition scheme is depicted in Figure 3 and aims at choosing wedges

```

CAD( $\mathcal{A}, \sigma_0$ )
1  $\sigma \leftarrow \sigma_0$ ;
2 while time limit unreached
3 do  $S \leftarrow \text{SELECTCUSTOMERS}(\mathcal{P}, \sigma)$ ;
4    $\mathcal{P}_o \leftarrow \text{EXTRACT}(S, \mathcal{P}, \sigma)$ ;
5    $\sigma_o^* \leftarrow \mathcal{A}(\mathcal{P}_o)$ ;
6    $\sigma \leftarrow \text{MERGE}(\mathcal{P}_o, \sigma_o^*, \sigma)$ ;
7 return  $\sigma$ 

EXTRACT( $S, \mathcal{P}, \sigma$ )
1  $R \leftarrow \{r \in \sigma \mid \exists c \in r : c \text{ lies in } S\}$ ;
2  $C_o \leftarrow \emptyset$ ;
3  $D_o^- \leftarrow \emptyset$ ;
4  $D_o^+ \leftarrow \emptyset$ ;
5 for  $r \in R$ 
6 do  $i \leftarrow \text{argmin}_{(c \in r \cap S)} a_c$ ;
7    $j \leftarrow \text{argmax}_{(c \in r \cap S)} a_c$ ;
8    $C_o \leftarrow C_o \cup \bigcup_{(c \in r) : a_i \leq a_c \leq a_j} c$ ;
9    $D_o^- \leftarrow D_o^- \cup \text{CONSTRUCTDEPARTUREDEPOT}(\text{pred}(i))$ ;
10   $D_o^+ \leftarrow D_o^+ \cup \text{CONSTRUCTARRIVALDEPOT}(\text{succ}(j))$ ;
11 return  $(C_o, D_o^+, D_o^-)$ 

CONSTRUCTARRIVALDEPOT( $p$ )
1  $d^- \leftarrow p$ ;
2  $[e_{d^-}, l_{d^-}] \leftarrow [\delta_p, \infty]$ ;
3  $q_{d^-} \leftarrow q(p)$ ;
4 return  $d^-$ ;

CONSTRUCTDEPARTUREDEPOT( $s$ )
1  $d^+ \leftarrow s$ ;
2  $[e_{d^+}, l_{d^+}] \leftarrow [0, z_s]$ ;
3  $q_{d^+} \leftarrow q(s) - q(\text{pred}(s))$ ;
4 return  $d^+$ ;

MERGE( $\mathcal{P}_o, \sigma_o^*, \sigma$ )
1 for  $c \in \mathcal{P}_o$ 
2 do  $\text{succ}(\sigma, \text{pred}(c)) \leftarrow c$ ;
3    $\text{pred}(\sigma, \text{succ}(c)) \leftarrow c$ ;
4    $\text{succ}(\sigma, c) \leftarrow \text{succ}(\sigma_o^*, c)$ ;
5    $\text{pred}(\sigma, c) \leftarrow \text{pred}(\sigma_o^*, c)$ ;
6 return  $\sigma$ ;

```

Fig. 2. The CAD Scheme

```

SELECTDECOMPOSITIONVASD( $\mathcal{P}, \sigma$ )
1  select  $\alpha \in [0, 359]$ ;
2  select  $\beta > \alpha$  such that the wedge  $W \leftarrow (\alpha, \beta)$ ;
3    (a) contains at least  $N$  customers;
4    (b) is the smallest wedge satisfying (a);
5   $\mathcal{V}_\lambda \leftarrow \{v \in \mathcal{V} \mid \exists c \in r_v : c \text{ lies in } W\}$ ;
6  return  $\bigcup_{v \in \mathcal{V}_\lambda} \text{cust}(r_v)$ ;

```

Fig. 3. The VASD Scheme for VRPTW Decouplings

```

SELECTDECOMPOSITIONCASD( $\mathcal{P}, \sigma$ )
1  select  $\alpha \in [0, 359]$ ;
2  select  $\beta > \alpha$  such that the wedge  $W \leftarrow (\alpha, \beta)$ ;
3    (a) contains at least  $N$  customers;
4    (b) is the smallest wedge satisfying (a);
5  return  $\bigcup c \text{ lies in } W$ ;

```

Fig. 4. The CASD Scheme for VRPTW Decouplings

producing roughly the same number N of customers. It first chooses the lower angle α of the wedge randomly (line 1). It then selects the upper angle β as the smallest angle greater than α producing the smallest wedge with at least N customers (lines 2–4). Finally, all customers of vehicles within in the wedge are included in the decomposition.

6.2 The CASD Scheme

We now present a customer-based spatial decomposition CASD that generalizes the VASD scheme. This generalization is especially important when considering problems (such as the class 2 problems of the extended Solomon benchmarks) where the vehicles serve many customers and can travel across many portions of the space. Under these conditions, VASD loses some of its locality as shown in Figure 1. In contrast, CASD algorithm preserves the spatial boundaries and improves the results of spatial decouplings on the class 2 extended Solomon benchmarks. Figure 4 gives the formalization of CASD which is a simplification of VASD. Figure 5 shows how the CASD scheme performs a decoupling from the same starting solution as Figure 1. The right hand picture shows all routes with decoupled customers, with the decoupled customers shown in red and the remaining ones in blue. It is interesting to compare this with Figure 1. CASD is clearly better at respecting spatial boundaries and allows customers of more vehicles to be considered in the decomposition.

6.3 The CATD Scheme

We now present a temporal instantiation (CATD) of the CAD scheme. The CATD scheme chooses random time slices and returns all of the customers that are served within that time slice. Figure 6 provides the implementation of this algorithm where lines 1–4 select a random slice that contains at least N customers. The mechanism for

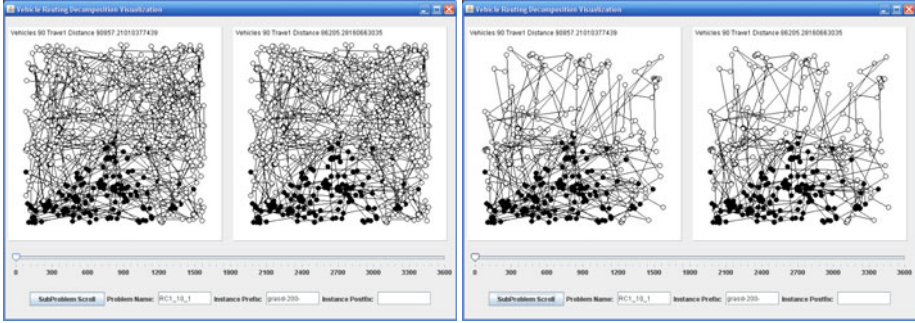


Fig. 5. The First Decoupling of CASD

SELECTDECOMPOSITIONCATD(\mathcal{P}, σ)

- 1 **select** $\alpha \in [0, l_{d \in \mathcal{D}}]$;
- 2 **select** $\beta > \alpha$ **such that** the time period $T = (\alpha, \beta)$;
- 3 (a) contains at least N customers;
- 4 (b) is the smallest time period satisfying (a);
- 5 **return** $\bigcup c$ served in T ;

Fig. 6. The CATD Scheme for VRPTW Decouplings

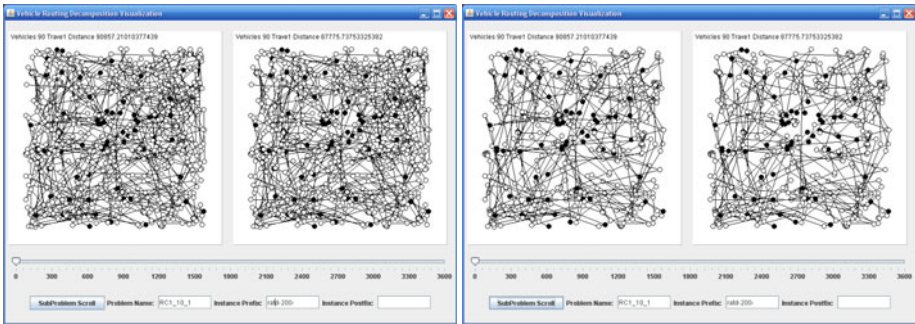


Fig. 7. The First Decoupling of CATD

choosing a time period is similar to that of CASD. First, α is chosen randomly from the interval $[0, l_{d \in \mathcal{D}}]$. β is then incremented from $\alpha + 1$ until the desired number of customers appear in the interval (or when $\beta = l_{d \in \mathcal{D}}$). Figure 7 demonstrates a decoupling based on the CATD scheme. Unlike prior decouplings, the temporal decoupling crosses most of the vehicles as seen by the number of routes included in the righthand side of the figure.

```

SELECTDECOMPOSITIONCARD( $\mathcal{P}, \sigma$ )
1  $S \leftarrow \emptyset$ ;
2 while  $|S| < N$ 
3 do select  $\alpha \in C \setminus S$ ;
4   select  $\beta \in C \setminus S$  such that  $route(\alpha) = route(\beta) \wedge \delta_\alpha < \delta_\beta$ ;
5    $S \leftarrow S \cup \bigcup c \in C$  such that  $route(c) = route(\alpha) \wedge \delta_\alpha \leq \delta_c \leq \delta_\beta$ ;
6 return  $S$ ;

```

Fig. 8. The CARD Scheme for VRPTW Decouplings

6.4 The CARD Scheme

This section describes a simple random decoupling scheme (CARD) used to provide a basis to evaluate the structured decoupling schemes described in the prior sections. Figure 8 shows the implementation. The scheme iterates by selecting random sequences of customers (lines 3–4) until the desired number of customers is achieved (line 2).

7 Experimental Results

This section presents the experimental results for the 1,000 customer extended Solomon benchmarks (www.top.sintef.no/vrp/benchmarks.html). The benchmarks contain a mix of loose and tight time windows and different types of spatial distributions. Recall that the difficulty in these problems, once two-stage algorithms are considered, is mostly in optimizing travel distances. Hence the experimental results mostly focus on this second stage, and uses a fixed solution with the minimal number of vehicles from the first phase. The experimental results use large neighborhood search (LNS) [29] for algorithm \mathcal{A} . LNS is one of the most effective algorithms for optimizing vehicle routing problems [29, 3, 2, 26, 24]; it also has the benefits of easily accommodating side constraints [3], which is important in practical implementations. The experiments report the solution quality under various time constraints (i.e., 2.5, 5, 10, and 15 minutes). Each reported result is the average of 50 runs on an AMD Athlon Dual Core Processor 3800.

For space reasons, we focus only on class R2. In general, the results on RC1 and R1 show that VASD(LNS) is the best implementation and produces significant improvements in solution quality under time constraints. In average, it produces improvements of 35%, 29%, 17%, and 6% over LNS when the time constraints require solutions to be found within 1, 2.5, 5, and 10 minutes respectively on RC1 problems. Both VASD and CASD outperform LNS on all RC1 and R1 instances and the results of CATD and CARD are good after the first 2.5 minutes. In general, good solutions to RC1 and R1 are characterized by vehicles serving very few customers in narrow regions, making spatial decompositions very natural. It is also important that the decomposition scheme provides highly competitive solutions when run for about an hour and improves some of the best-known solutions on these benchmarks.

Table 1. R2 Solution Quality Under Time Constraints

BK UB	R2_10.1	R2_10.2	R2_10.3	R2_10.4	R2_10.5	R2_10.6	R2_10.7	R2_10.8	R2_10.9	R2_10.10	Avg
LNS (1)	56336.2	43864.4	42620.2	33281.9	47352.4	40907.5	38056.6	29516.6	44540.9	40973.8	
VASD (1)	67937.5	48391.3	47151.0	29349.6	59075.1	49561.9	38104.1	26981.8	55860.8	47982.2	
%Impr.	-20.6	-10.3	-10.6	11.8	-24.8	-21.2	-0.1	8.6	-25.4	-17.1	-11.0
CASD (1)	68108.7	50337.8	47611.1	28388.1	56962.9	50430.3	37210.3	25543.2	56308.1	51254.3	
%Impr.	-20.9	-14.8	-11.7	14.7	-20.3	-23.3	2.2	13.5	-26.4	-25.1	-11.2
CATD (1)	51346.6	42352.3	41823.3	34448.2	46275.7	43460.7	38282.0	31186.8	42321.8	42122.2	
%Impr.	8.9	3.4	1.9	-3.5	2.3	-6.2	-0.6	-5.7	5.0	-2.8	0.3
CARD (1)	76915.0	63039.1	57772.6	40365.4	59084.6	60321.9	48678.5	35275.7	66420.1	62966.8	
%Impr.	-36.5	-43.7	-35.6	-21.3	-24.8	-47.5	-27.9	-19.5	-49.1	-53.7	-36.0
LNS (2.5)	53667.5	41260.3	37907.6	30007.5	44941.5	38028.4	33939.7	26921.2	42134.0	38351.7	
VASD (2.5)	58759.6	41955.8	38316.4	24632.8	49847.5	38975.1	32055.8	23029.9	46152.3	40896.1	
%Impr.	-9.5	-1.7	-1.1	17.9	-10.9	-2.5	5.6	14.5	-9.5	-6.6	-0.4
CASD (2.5)	54423.3	40426.1	33387.2	22717.8	46063.0	38462.0	29460.6	20837.6	43235.2	39663.2	
%Impr.	-1.4	2.0	11.9	24.3	-2.5	-1.1	13.2	22.6	-2.6	-3.4	6.3
CATD (2.5)	46203.4	38061.9	33749.0	28799.4	40220.7	36499.3	32848.3	26997.1	37653.3	34791.7	
%Impr.	13.9	7.8	11.0	4.0	10.5	4.0	3.2	-0.3	10.6	9.3	7.4
CARD (2.5)	65820.0	50880.9	43792.2	31651.3	56636.9	47119.1	37598.2	25336.4	54375.1	50583.8	
%Impr.	-22.6	-23.3	-15.5	-5.5	-26.0	-23.9	-10.8	5.9	-29.1	-31.9	-18.3
LNS (5)	51877.8	39871.7	34873.2	27549.9	43616.4	36400.2	31500.3	25323.0	40647.4	37109.6	
VASD (5)	54743.7	40546.3	34540.3	22899.2	46174.3	36959.9	30188.5	21775.7	42417.0	38351.2	
%Impr.	-5.5	-1.7	1.0	16.9	-5.9	-1.5	4.2	14.0	-4.4	-3.3	1.4
CASD (5)	49454.3	38194.8	30138.7	21578.2	42203.5	34796.8	27451.5	20837.6	38577.2	35847.8	
%Impr.	4.9	4.4	15.7	27.7	3.3	4.6	14.7	21.5	5.4	3.5	10.6
CATD (5)	44633.9	36339.7	31647.7	26463.1	39040.4	34816.3	29354.2	25357.9	36014.8	33517.4	
%Impr.	14.0	8.9	9.2	3.9	10.5	4.4	6.8	-0.1	11.4	9.7	7.9
CARD (5)	58595.0	44269.4	37808.6	27458.0	49889.9	40801.7	32854.8	23217.5	47379.4	43606.2	
%Impr.	-5.5	-1.7	1.0	16.9	-5.9	-1.5	4.2	14.0	-4.4	-3.3	1.4
LNS (10)	50763.2	38737.0	34873.2	25195.6	42848.5	35342.0	29752.8	23665.7	39802.5	36378.8	
VASD (10)	51950.6	39427.7	32426.1	22185.2	44327.2	35842.8	29264.1	21164.4	40519.9	37099.5	
%Impr.	-2.3	-1.8	7.0	11.9	-3.5	-1.4	1.6	10.6	-1.8	-2.0	1.8
CASD (10)	47371.3	37343.2	28991.6	21010.5	40890.2	33852.4	26566.4	20290.5	37112.5	34632.8	
%Impr.	6.7	3.6	16.9	16.6	4.6	4.2	10.7	14.3	6.8	4.8	8.9
CATD (10)	44172.2	36339.7	31358.7	25469.9	38445.3	33830.0	29354.2	24371.9	35221.8	32786.1	
%Impr.	13.0	6.2	10.1	-1.1	10.3	4.3	1.3	-3.0	11.5	9.9	6.2
CARD (10)	52845.8	40408.6	33549.5	24331.5	45462.6	36987.2	29852.5	23217.5	42537.3	38638.1	
%Impr.	-4.1	-4.3	3.8	3.4	-6.1	-4.7	-0.3	1.9	-6.9	-6.2	-2.3

Benefits of CAD. Table 1 describes the solution quality under various time constraints for LNS and various instantiations of CAD(LNS) on R2 problems. Each column describes a R2 instance with 1,000 customers and the best-known number of vehicles. The clusters of rows consider various time constraints: 1, 2.5, 5, and 10 minutes. The row *BK* specifies the travel distance of the best known solution (prior to this research). The rows *%Impr* describes the improvement in solution quality of CAD(LNS) with respect to LNS. CAD(LNS) is run with $N = 200$, i.e., the decomposition must contain at least 200 customers.

It is interesting to observe that Table 1 provides very different conclusions than the results on classes RC1 and R1. High-quality solutions to R2 problems are characterized by fewer vehicles serving many more customers over wide temporal regions. This puts VASD at a disadvantage as decompositions typically violate the natural spatial boundaries of the wedge due to the need to include all customers of vehicles. This is best illustrated by the 5 minute results, when the CASD scheme vastly outperforms VASD. After 2.5, 5, and 10 minutes, CASD produces average improvements of 6.3%, 10.6%, and 8.9% over LNS, while VASD degrades the performance after 2.5 minutes and

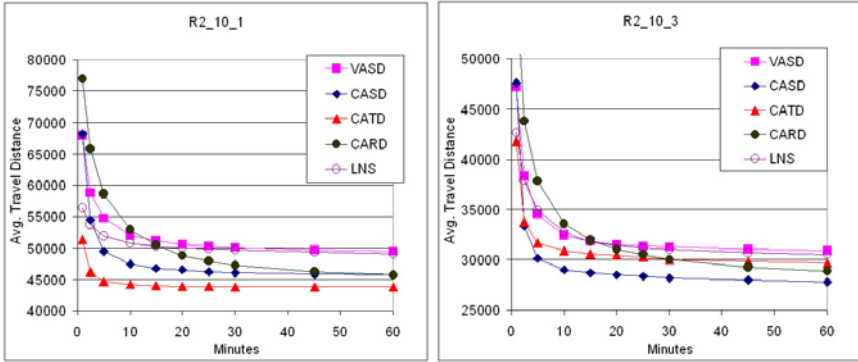


Fig. 9. Benefits of CAD on R2_10_1 and R2_10_3

produces improvements of 1.4% and 1.6% for 5 and 10 minutes. On some benchmarks, CASD produces more than 10% over LNS. Interestingly, CATD produces excellent results on class R2 and produces average improvements of 0.3%, 7.5%, 7.9%, and 6.2% after 1, 2.5, 5, and 10 minutes. Moreover, it significantly outperforms other decompositions on several benchmarks where it can produce improvements up to 14%. On closer inspection, CATD performs very well on those problems whose customers have narrow time windows. The explanation for this behavior is interesting: when a customer has a wide time window, it can be served early or late. If it is initially served early when it should be served late, it is impossible to find a solution that moves the customer to a later time period, unless every intermediate temporal decoupling provides an improving solution. On problems with customers with narrow time windows, the problem structure itself enforces the correct temporal locations of the customers, making a temporal decomposition very natural.

Figure 9 depicts the typical behavior of LNS and CAD(LNS) on two benchmarks in the R2 class. In the left graph, the R2 problem has narrow time windows and CATD is clearly the best, further demonstrating the natural benefits of this decomposition when customers have narrow time windows. It also shows the limitations of the VASD approach under the conditions of class 2 problems. The right part of the figure shows results on a class 2 problem with wide time windows. Here we see a reversal of the effectiveness of CATD where CASD is clearly better. Note also that CASD(LNS) and CATD(LNS) still dominates LNS when both algorithms run for an hour.

Overall, these results clearly show the benefits of customer-based decompositions and the complementary between spatial and temporal decompositions.

Hybrid Implementations. To exploit this complementarity, We also considered some hybrid approaches between CASD and CATD to determine if a single approach would perform well on all instances (for example good on both R2_10_1 and R2_10_3). Two hybrids worked quite well. The first hybrid chooses to either follow a CATD decoupling or a CASD decoupling randomly at each iteration. The second hybrid creates a decoupling at each iteration that contains $N/2$ customers from a CATD selection and

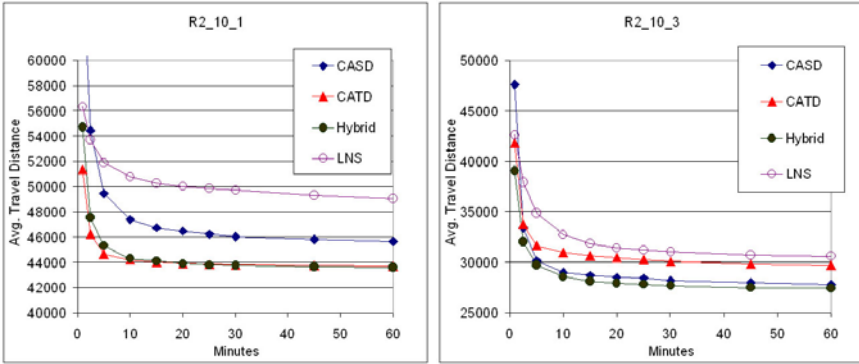


Fig. 10. Benefits of Hybrid Approaches

$N/2$ from CASD selection. Both schemes generated very consistent results on all problems, in general being within 1% of the best CASD or CATD result on each problem. This indicates that when problem structure is unknown or varied, a hybrid approach may produce the best results. Figure 10 demonstrates how the first hybrid approach smoothes out performance.

8 Related Work

There are literally hundreds of papers discussing vehicle routing problems and their variations and it is beyond the scope of this paper to provide a comprehensive literature review. The reader is invited to see [8,9,11,16,27,25] for recent surveys. Almost all papers focus on problems of relatively small size which, as mentioned earlier, are already extremely difficult. Unfortunately, many of the proposed techniques do not scale well and some recent papers specifically address large-scale problems. We now focus attention on recent work that have considered decomposition ideas.

Decomposition comes in many different varieties in literature. In some papers, like [5,6], decomposition focuses on decomposing the search strategy space (as opposed to problem structure). Related to this idea is the view of decomposition across attributes (variables) of the problems. Multi-stage approaches such as [15,21,18,10,7] can be classified in this way (i.e., first minimizing the number vehicles required and then minimizing the travel distance). [12] suggests a general framework for breaking problems across attribute boundaries using evolutionary algorithms. The different sub-problems communicate results via population exchanges. The framework is tested on the VRPTW. The key difference between attribute decomposition and CAD is that our approach retains information about the entire problem and simplifies the problem by decreasing their scale.

Recent and concurrent work has focused on dividing the problem into smaller sub-problems across structural boundaries that is very much in the spirit of VASD. [20] presents a deterministic hierarchical decomposition scheme for evolutionary algorithms. The VRPTW spatial region is divided into rectangles, defining sub problems that are

solved independently. The rectangles are recursively merged into larger subproblems which rely on the smaller problems as starting solutions for the larger subproblems. [1] introduces spatial-based decomposition ideas in a genetic algorithm. Their approach randomly applies the evolutionary operations to either the whole problem or spatially defined sub regions. [23][22] presents some interesting spatial decomposition approaches based on clustering (POPMUSIC). At a high level, POPMUSIC iteratively chooses routes and creates subproblems based on *nearness* to that route (different approaches to defining nearness are explored). The algorithm iterates until it has created subproblems on all routes without improvement. Finally, the work of [13] proposes a decoupling scheme for the air-taxi problem based on spatial boundaries.

In many ways, all of these approach can be viewed as variations of VASD. *The key difference between these approaches and our framework is that they decompose problems based on routes as opposed to customers.* This makes the merging of solutions from the subproblems to the global problem easy. However, by structuring the decompositions on a customer basis, we are able to create subproblems *within* routes, a property that is very important when routes cross multiple spatial and temporal boundaries. But is important to note that this related work also supports our claim that decomposition improves algorithm performance.

It is useful to contrast the deconstruction steps of LNS ([29][26][28][24]) and the CAD scheme. In LNS, the basic step consists of removing related customers (often based on spatial or temporal relationships) from a plan σ and to reinsert them in σ using an optimization algorithm. The CAD scheme can also be thought of as removing related customers with two fundamental differences: 1) *the removed customers defines a VRPTW subproblem of (significantly) smaller size which can solved independently* and 2) *Subproblems restrict neighborhood explorations to being within the decomposition itself.* This is critical for finding high-quality solution quickly. Obviously, the two approaches are synergetic since our results are obtained using CAD(LNS).

Finally, it is useful to relate CAD to the approach in [17] which impose specific temporal constraints to obtain decouplings. CAD uses spatial and temporal decouplings that constrain specific subsets of customers to be served by designated vehicles. Moreover, the use of decoupling is fundamentally different. *The idea is to iteratively obtain new decouplings to optimize an existing plan by re-optimizing subproblems.* This use of decouplings also contrast with traditional decomposition techniques in constraint satisfaction [14].

9 Conclusion

This paper reconsidered the adaptive decomposition framework to quickly find high-quality solutions to large-scale vehicle routing problems with time windows. Earlier work had focused vehicle-based decompositions that partition the vehicles across the subproblems which makes it easy to define the subproblems and merge their solutions. Although vehicle-based spatial decompositions are very effective on classes R1 and RC1 of the extended Solomon benchmarks, the paper identified some of their limitations and, in particular, the difficulty in adapting them to temporal decompositions. This paper then proposed customer-based decompositions which generalize vehicle-based

decouplings and allow for focused spatial and temporal decompositions. Experimental results on class R2 of the extended Solomon benchmarks demonstrated the benefits of the customer-based adaptive decomposition scheme and its spatial, temporal, and hybrid instantiations. In particular, the results show significant benefits over the use of large neighborhood search and vehicle-based spatial decompositions. For instance, customer-based temporal decompositions yield an average improvement of 7.4% over LNS after 2.5 minutes, while the vehicle-based spatial decomposition degrades the performance by 0.4% in average. Similarly, customer-based spatial decompositions yield an average improvement of 10.6% over LNS after 5 minutes, while the vehicle-based spatial decomposition improves the performance by only 1.4% in average. The complementary between spatial and temporal decompositions was also highlighted and hybridizations were shown to be particularly effective in producing robust results across all benchmarks. An intriguing future research direction is to determine whether the decomposition can be chosen automatically from the instance structure.

References

1. Alvarenga, G.B., Mateus, G.R., de Tomi, G.: A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers & Operations Research* 34, 1561–1584 (2007)
2. Bent, R., Van Hentenryck, P.: A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. *Transportation Science* 38(4), 515–530 (2004)
3. Bent, R., Van Hentenryck, P.: A Two-Stage Hybrid Algorithm for Pickup and Delivery Vehicle Routing Problems with Time Windows. *Computers and Operations Research* 33(4), 875–893 (2006)
4. Bent, R., Van Hentenryck, P.: Randomized Adaptive Spatial Decoupling For Large-Scale Vehicle Routing with Time Windows. In: *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI)*, Vancouver, Canada (2007)
5. Le Bouthillier, A., Crainic, T.: A Cooperative Parallel Meta-Heuristic for the Vehicle Routing Problem with Time Windows. *Computers and Operations Research* 32, 1685–1708 (2005)
6. Le Bouthillier, A., Crainic, T., Kropf, P.: A Guided Cooperative Search for the Vehicle Routing Problem with Time Windows. *IEEE Intelligent Systems* 20(4), 36–42 (2005)
7. Braysy, O.: A Reactive Variable Neighborhood Search for the Vehicle Routing Problem with Time Windows. *INFORMS Journal on Computing* 15(4), 347–368 (2003)
8. Braysy, O., Gendreau, M.: Vehicle Routing Problems with Time Windows, Part i: Route Construction and Local Search Algorithms. *Transportation Science* 39, 104–118 (2005)
9. Braysy, O., Gendreau, M.: Vehicle Routing Problems with Time Windows, Part ii: Metaheuristics. *Transportation Science* 39, 119–139 (2005)
10. Braysy, O., Hasle, G., Dullaert, W.: A Multi-Start Local Search for the Vehicle Routing Problem with Time Windows. *European Journal of Operational Research* 159(3), 586–605 (2004)
11. Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M., Soumis, F.: The VRP with Time Windows. In: *The Vehicle Routing Problem: SIAM Monographs on Discrete Mathematics and Applications*, pp. 157–194 (2001)
12. Crainic, T.G., Crisan, G.C., Gendreau, M., Lahrichi, N., Rei, W.: A concurrent evolutionary approach for rich combinatorial optimization. In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (GECCO 2009)*, pp. 2017–2022. ACM, New York (2009)

13. Espinoza, D., Garcia, R., Goycoolea, M., Nemhauser, G.L., Savelsbergh, M.: Per-Seat, On-Demand Air Transportation Part II: Parallel Local Search. *Transportation Science* 42(3), 279–291 (2008)
14. Dechter, R.: *Constraint Processing*. Morgan Kaufmann, San Francisco (2003)
15. Gehring, H., Homberger, J.: A Parallel Two-phase Metaheuristic for Routing Problems with Time Windows. *Asia-Pacific Journal of Operational Research* 18, 35–47 (2001)
16. Golden, B., Raghavan, S., Wasil, E.: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, Heidelberg (2008)
17. Hunsberger, L.: Algorithms for a Temporal Decoupling Problem in Multi-Agent Planning. In: *Proceedings of the Eighteenth American Conference on Artificial Intelligence (AAAI)*, Edmonton, Canada, pp. 468–475 (2002)
18. Lim, A., Zhang, X.: A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing* 19(3), 443–457 (2007)
19. Mester, D., Braysy, O.: Active Guided Evolution Strategies for Large Scale Vehicle Routing Problems with Time Windows. *Computers and Operations Research* 32, 1593–1614 (2005)
20. Mester, D., Braysy, O., Dullaert, W.: A multi-parametric evolution strategies algorithm for vehicle routing problems. *Expert Systems with Applications* 32, 508–517 (2007)
21. Nagata, Y., Baysy, O., Dullaert, W.: A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers and Operations Research* 37(4), 724–737 (2010)
22. Ostertag, A., Doerner, K., Hartl, R., Taillard, E., Waelti, P.: POPMUSIC for a real-world large-scale vehicle routing problem with time windows. *Journal of the Operational Research Society* 60(7), 934–943 (2009)
23. Ostertag, A., Doerner, K.F., Hartl, R.F.: A variable neighborhood search integrated in the popmusic framework for solving large scale vehicle routing problems. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (eds.) *HM 2008*. LNCS, vol. 5296, pp. 29–42. Springer, Heidelberg (2008)
24. Pisinger, D., Ropke, S.: Large neighborhood search. In: Potvin, J.-Y., Gendreau, M. (eds.) *Handbook of Metaheuristics*. Springer, Heidelberg (2009)
25. Potvin, J.-Y.: A review of bio-inspired algorithms for vehicle routing. In: Pereira, F.B., Tavares, J. (eds.) *SCI*, ch. 1, pp. 1–34. Springer, Heidelberg (2008)
26. Prescott-Gagnon, E., Desaulniers, G., Rousseau, L.-M.: A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 1–15 (to appear)
27. Repoussis, P.P., Tarantilis, C.D., Ioannou, G.: Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions on Evolutionary Computation* 13(3), 624–647 (2009)
28. Rousseau, L.-M., Gendreau, M., Pesant, G.: Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows. *Journal of Heuristics* 8(1), 43–58 (2002)
29. Shaw, P.: Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: Maher, M.J., Puget, J.-F. (eds.) *CP 1998*. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)

Decomposition of the NVALUE Constraint

Christian Bessiere¹, George Katsirelos², Nina Narodytska³,
Claude-Guy Quimper⁴, and Toby Walsh³

¹ LIRMM, CNRS, Montpellier
bessiere@lirmm.fr

² CRIL-CNRS, Lens
gkatsi@gmail.com

³ NICTA and University of NSW, Sydney, Australia
{nina.narodytska, toby.walsh}@nicta.com.au

⁴ Université Laval
cquimper@gmail.com

Abstract. We study decompositions of the global NVALUE constraint. Our main contribution is theoretical: we show that there are propagators for global constraints like NVALUE which decomposition can simulate with the same time complexity but with a much greater space complexity. This suggests that the benefit of a global propagator may often not be in saving time but in saving space. Our other theoretical contribution is to show for the first time that range consistency can be enforced on NVALUE with the same worst-case time complexity as bound consistency. Finally, the decompositions we study are readily encoded as linear inequalities. We are therefore able to use them in integer linear programs.

1 Introduction

Global constraints are one of the distinguishing features of constraint programming. They capture common modelling patterns and have associated efficient propagators for pruning the search space. For example, ALL-DIFFERENT is one of the best known global constraints that has proven useful in the modelling and solving of many real world problems. A number of efficient algorithms have been proposed to propagate the ALL-DIFFERENT constraint (e.g. [1,2,3]). Whilst there is little debate that ALL-DIFFERENT is a global constraint, the formal definition of a global constraint is more difficult to pin down. One property often associated with global constraints is that they cannot be decomposed into simpler constraints without impacting either the pruning or the efficiency of propagation [4]. Recently progress has been made on the theoretical problem of understanding what is and isn't a global constraint. In particular, whilst a bound consistency propagator for the ALL-DIFFERENT constraint can be effectively simulated with a simple decomposition [5], circuit complexity lower bounds have been used to prove that a domain consistency propagator for ALL-DIFFERENT cannot be polynomially simulated by a simple decomposition [6].

In this paper, we turn to a strict generalization of the ALL-DIFFERENT constraint. NVALUE counts the number of values used by a set of variables; the ALL-DIFFERENT constraint ensures that this count equals the cardinality of the set. From a theoretical

perspective, the NVALUE constraint is significantly more difficult to propagate than the ALL-DIFFERENT constraint since enforcing domain consistency is known to be NP-hard [7]. Moreover, as NVALUE is a generalization of ALL-DIFFERENT, there exists no polynomial sized decomposition of NVALUE which achieves domain consistency [6]. Nevertheless, we show that decomposition can simulate the polynomial time algorithm for enforcing bound consistency on NVALUE but with a significant space complexity. We also prove, for the first time, that range consistency on NVALUE can be enforced in the same worst case time complexity as bound consistency. This contrasts with the ALL-DIFFERENT constraint where range consistency takes $O(n^2)$ time [2] but bound consistency takes just $O(n \log n)$ time [3].

The main value of these decompositions is theoretical as their space complexity is equal to their worst case time complexity. When domains are large, this space complexity may be prohibitive. In the conclusion, we argue why it appears somewhat inevitable that the space complexity is equal to the worst case time complexity. These results suggest new insight into what is and isn't a global constraint: a global constraint either provides more pruning than any polynomial sized decomposition or provides the same pruning but with lower space complexity. There are several other theoretical reasons why the decompositions studied here are interesting. First, it is technically interesting that a complex propagation algorithm like the bound consistency propagator for NVALUE can be simulated by a simple decomposition. Second, these decompositions can be readily encoded as linear inequalities and used in linear programs. In fact, we will report experiments using both constraint and integer linear programming with these decompositions. Since global constraints are one of the key differentiators between constraint and integer programming, these decompositions provide us with another tool to explore the interface between constraint and integer programming. Third, the decompositions give insights into how we might add nogood learning to a NVALUE propagator.

2 Background

A constraint satisfaction problem (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints. We use capitals for variables and lower case for values. We assume values are taken from the set 1 to d . We write $dom(X_i)$ for the domain of possible values for X_i , $min(X_i)$ for the smallest value in $dom(X_i)$, $max(X_i)$ for the greatest, and $range(X_i)$ for the interval $[min(X_i), max(X_i)]$. Constraint solvers typically use backtracking search to explore the space of partial assignments. After each assignment, propagation algorithms prune the search space by enforcing local consistency properties like domain, range or bound consistency. A constraint is *domain consistent (DC)* iff when a variable is assigned any of the values in its domain, there exist compatible values in the domains of all the other variables of the constraint. Such an assignment is called a *support*. A CSP is domain consistent iff every constraint is domain consistent. A constraint is *disentailed* iff there is no possible support. A propagator which enforces domain consistency will detect disentanglement, but a propagator that detects just disentanglement will not enforce domain consistency. A constraint is *range consistent (RC)* iff, when a variable is assigned any of the values in its domain, there exist compatible values between the minimum and maximum domain

value for all the other variables of the constraint. Such an assignment is called a *bound support*. A constraint is *bound consistent (BC)* iff the minimum and maximum value of every variable of the constraint belong to a bound support. A CSP is bound consistent iff every constraint is bound consistent. We compute the total amortized cost of enforcing a local consistency down an entire branch of the search tree. This captures the incremental cost of propagation. Finally, we will assume that a propagator is invoked at most once for each domain change and that the solver uses an optimal propagator to enforce BC on sum and channeling constraints. Such assumptions hold for modern solvers like Gecode and Ilog Solver. However, we make no assumption about the order of invocation of the constraints in a decomposition. The upper bounds we give hold *regardless* of the order in which constraints are processed.

A *global constraint* is one in which the arity of the constraint n is a parameter. A *decomposition* of a global constraint is a CSP involving the n variables of the global constraint (and possibly others), involving only constraints with fixed arity (no global constraint) or constraints that are themselves decomposable, such that the size of the CSP is polynomial in the sum of the sizes of the domains of the n original variables, and such that the projection of its solutions on those n variables corresponds to the solutions of the global constraint. A useful notion is algorithmic globality [4]. Informally, given a local consistency property, a global constraint is algorithmically global if there is no decomposition on which this local consistency is achieved in the same time and space complexity. We suggest here two refinements of this notion of algorithmic globality. First, we will separate the space and time complexity. That is, given a local consistency property, a global constraint is algorithmically global with respect to time (space) if there is no decomposition on which this local consistency is achieved in the same time (space) complexity. Second, unlike [4], we consider decompositions that may introduce new variables. Our results will show that, when we introduce new variables, NVALUE is not algorithmically global with respect to time but *is* global with respect to space.

3 NVALUE Constraint

Pachet and Roy first proposed the NVALUE constraint [8]. Formally $NVALUE([X_1, \dots, X_n], N)$ ensures that $N = |\{X_i \mid 1 \leq i \leq n\}|$. This generalizes several other global constraints including ALL-DIFFERENT (which ensures that the number of values taken by a set of variables equals the cardinality of the set) and NOT-ALL-EQUAL (which ensures a set of variables take more than one value). Enforcing domain consistency on the NVALUE constraint is NP-hard (Theorem 3 in [7]) even when N is fixed (Theorem 2 in [9]). In fact, just computing the lower bound on N is NP-hard (Theorem 3 in [10]). In addition, enforcing domain consistency on the NVALUE constraint is not fixed parameter tractable since it is $W[2]$ -complete [11]. However, several polynomial propagation algorithms have been proposed that achieve bound consistency and some closely related levels of local consistency [12,9,13].

3.1 Simple Decomposition

Global constraints can often be decomposed into simpler, more primitive and small arity constraints. For example, the ALL-DIFFERENT constraint can be decomposed into

a quadratic number of binary inequalities. However, such decomposition often hinders propagation and can have a significant impact on the solver’s ability to find solutions [14]. We can decompose the NVALUE constraint by introducing 0/1 variables to represent which values are used and posting a sum constraint on these introduced variables:

$$X_i = j \rightarrow B_j = 1 \quad \forall 1 \leq i \leq n, 1 \leq j \leq d \tag{1}$$

$$B_j = 1 \rightarrow \bigvee_{i=1}^n X_i = j \quad \forall 1 \leq j \leq d \tag{2}$$

$$\sum_{j=1}^d B_j = N \tag{3}$$

Note that constraint 3 is not a fixed arity constraint, but can itself be decomposed to ternary sums without hindering bound propagation. Unfortunately, this simple decomposition hinders propagation. It can be BC whereas BC on the corresponding NVALUE constraint detects disentanglement.

Theorem 1. *BC on NVALUE is stronger than BC on its decomposition into (1) to (3).*

Proof: Clearly BC on NVALUE is at least as strong as BC on the decomposition. To show strictness, consider $X_1 \in \{1, 2\}$, $X_2 \in \{3, 4\}$, $B_j \in \{0, 1\}$ for $1 \leq j \leq 4$, and $N = 1$. Constraints (1) to (3) are BC. However, the corresponding NVALUE constraint has no bound support and thus enforcing BC on it detects disentanglement. □

We observe that enforcing DC instead of BC on constraints (1) to (3) in the example of the proof above still does not prune any value. To decompose NVALUE without hindering propagation, we must look to more complex decompositions.

3.2 Decomposition into ATMOSTNVALUE and ATLEASTNVALUE

Our first step in decomposing the NVALUE constraint is to split it into two parts: an ATMOSTNVALUE and an ATLEASTNVALUE constraint. ATLEASTNVALUE($[X_1, \dots, X_n], N$) holds iff $N \leq |\{X_i | 1 \leq i \leq n\}|$ whilst ATMOSTNVALUE($[X_1, \dots, X_n], N$) holds iff $|\{X_i | 1 \leq i \leq n\}| \leq N$.

Running Example. *Consider a NVALUE constraint over the following variables and values:*

	1	2	3	4	5
X_1	*	*	*	*	*
X_2		*			
X_3	*	*	*		
X_4		*	*		
X_5		*	*		
N	*	*		*	

Suppose we decompose this into an ATMOSTNVALUE and an ATLEASTNVALUE constraint. Consider the ATLEASTNVALUE constraint. The 5 variables can take at most 4 different values because X_2, X_3, X_4 , and X_5 can only take values 2, 3 and 4. Hence, there is no bound support for $N = 5$. Enforcing BC on the ATLEASTNVALUE constraint therefore prunes $N = 5$. Consider now the ATMOSTNVALUE constraint. Since X_2 and X_4 guarantee that we take at least 2 different values, there is no bound support for $N = 1$. Hence enforcing BC on an ATMOSTNVALUE constraint prunes $N = 1$.

$X_1 = 1, 3$ or 5 , or $X_5 = 3$ then any complete assignment uses at least 3 different values. Hence there is also no bound support for these assignments. Pruning these values gives bound consistent domains for the original NVALUE constraint:

	1	2	3	4	5
X_1		*			
X_2		*			
X_3		*	*	*	
X_4				*	
X_5					*
N					*

To show that decomposing the NVALUE constraint into these two parts does not hinder propagation in general, we will use the following lemma. Given an assignment S of values, $card(S)$ denotes the number of distinct values in S . Given a vector of variables $X = X_1 \dots X_n$, $card_{\uparrow}(X) = \max\{card(S) \mid S \in \prod_{X_i \in X} range(X_i)\}$ and $card_{\downarrow}(X) = \min\{card(S) \mid S \in \prod_{X_i \in X} range(X_i)\}$.

Lemma 1 (adapted from [13]). Consider $NVALUE([X_1, \dots, X_n], N)$. If $dom(N) \subseteq [card_{\downarrow}(X), card_{\uparrow}(X)]$, then the bounds of N have bound supports.

Proof: Let S_{min} be an assignment of X in $\prod_{X_i \in X} range(X_i)$ with $card(S_{min}) = card_{\downarrow}(X)$ and S_{max} be an assignment of X in $\prod_{X_i \in X} range(X_i)$ with $card(S_{max}) = card_{\uparrow}(X)$. Consider the sequence $S_{min} = S_0, S_1, \dots, S_n = S_{max}$ where S_{k+1} is the same as S_k except that X_{k+1} has been assigned its value in S_{max} instead of its value in S_{min} . $|card(S_{k+1}) - card(S_k)| \leq 1$ because they only differ on X_{k+1} . Hence, for any $p \in [card_{\downarrow}(X), card_{\uparrow}(X)]$, there exists $k \in 1..n$ with $card(S_k) = p$. Thus, (S_k, p) is a bound support for p on $NVALUE([X_1, \dots, X_n], N)$. Therefore, $min(N)$ and $max(N)$ have a bound support. \square

We now prove that decomposing the NVALUE constraint into ATMOSTNVALUE and ATLEASTNVALUE constraints does not hinder pruning when enforcing BC.

Theorem 2. BC on $NVALUE([X_1, \dots, X_n], N)$ is equivalent to BC on $ATMOSTNVALUE([X_1, \dots, X_n], N)$ and on $ATLEASTNVALUE([X_1, \dots, X_n], N)$.

Proof: Suppose the ATMOSTNVALUE and ATLEASTNVALUE constraints are BC. The ATMOSTNVALUE constraint guarantees that $card_{\downarrow}(X) \leq min(N)$ and the ATLEASTNVALUE constraint guarantees that $card_{\uparrow}(X) \geq max(N)$. Therefore, $dom(N) \in [card_{\downarrow}(X), card_{\uparrow}(X)]$. By Lemma 1 the variable N is bound consistent.

Consider a variable/bound value pair $X_i = b$. Let (S_{least}^b, p_1) be a bound support of $X_i = b$ in the ATLEASTNVALUE constraint and (S_{most}^b, p_2) be a bound support of $X_i = b$ in the ATMOSTNVALUE constraint. We have $card(S_{least}^b) \geq p_1$ and $card(S_{most}^b) \leq p_2$ by definition of ATLEASTNVALUE and ATMOSTNVALUE. Consider the sequence $S_{least}^b = S_0^b, S_1^b, \dots, S_n^b = S_{most}^b$ where S_{k+1}^b is the same as S_k^b except that X_{k+1} has been assigned its value in S_{most}^b instead of its value in S_{least}^b . $|card(S_{k+1}^b) - card(S_k^b)| \leq 1$ because they only differ on X_{k+1} . Hence, there exists $k \in 1..n$ with $min(p_1, p_2) \leq card(S_k^b) \leq max(p_1, p_2)$. We know that p_1 and p_2 belong to $range(N)$ because they belong to bound supports. Thus, $card(S_k^b) \in range(N)$ and $(S_k^b, card(S_k^b))$ is a bound support for $X_i = b$ on $NVALUE([X_1, \dots, X_n], N)$. \square

When enforcing domain consistency, Bessiere *et al.* [13] noted that decomposing the NVALUE constraint into ATMOSTNVALUE and ATLEASTNVALUE constraints does hinder propagation, but only when $\text{dom}(N)$ contains just $\text{card}_\downarrow(X)$ and $\text{card}_\uparrow(X)$ and there is a gap in the domain in-between (see Theorem 1 in [13] and the discussion that follows). When enforcing BC, any such gap in the domain for N is ignored.

4 ATMOSTNVALUE Constraint

We now give a decomposition for the ATMOSTNVALUE constraint which does not hinder bound consistency propagation. To decompose the ATMOSTNVALUE constraint, we introduce 0/1 variables, A_{ilu} to represent whether X_i uses a value in the interval $[l, u]$, and “pyramid” variables, M_{lu} with domains $[0, \min(u - l + 1, n)]$ which count the number of values taken inside the interval $[l, u]$. To constrain these introduced variables, we post the following constraints:

$$A_{ilu} = 1 \iff X_i \in [l, u] \quad \forall 1 \leq i \leq n, 1 \leq l \leq u \leq d \quad (4)$$

$$A_{ilu} \leq M_{lu} \quad \forall 1 \leq i \leq n, 1 \leq l \leq u \leq d \quad (5)$$

$$M_{1u} = M_{1k} + M_{(k+1)u} \quad \forall 1 \leq k < u \leq d \quad (6)$$

$$M_{1d} \leq N \quad (7)$$

Running Example. Consider the decomposition of an ATMOSTNVALUE constraint over the following variables and values:

	1	2	3	4	5
X_1	*	*	*	*	*
X_2	*				
X_3	*	*	*		
X_4				*	
X_5		*	*		
N	*	*			

Observe that we consider that value 5 for N has already been pruned by ATLEASTNVALUE, as will be shown in next sections. Bound consistency reasoning on the decomposition will make the following inferences. As $X_2 = 2$, from (4) we get $A_{222} = 1$. Hence by (5), $M_{22} = 1$. Similarly, as $X_4 = 4$, we get $A_{444} = 1$ and $M_{44} = 1$. Now $N \in \{1, 2\}$. By (7) and (6), $M_{15} \leq N$, $M_{15} = M_{14} + M_{55}$, $M_{14} = M_{13} + M_{44}$, $M_{13} = M_{12} + M_{33}$, $M_{12} = M_{11} + M_{22}$. Since $M_{22} = M_{44} = 1$, we deduce that $N > 1$ and hence $N = 2$. This gives $M_{11} = M_{33} = M_{55} = 0$. By (5), $A_{111} = A_{133} = A_{155} = A_{533} = 0$. Finally, from (4), we get $X_1 = 2$ and $X_5 = 3$. This gives us bound consistent domains for the ATMOSTNVALUE constraint.

We now prove that this decomposition does not hinder propagation in general.

Theorem 3. BC on constraints (4) to (7) is equivalent to BC on ATMOSTNVALUE $([X_1, \dots, X_n], N)$, and takes $O(nd^3)$ time to enforce down the branch of the search tree.

Proof: First note that changing the domains of the X variables cannot affect the upper bound of N by the ATMOSTNVALUE constraint and, conversely, changing the lower bound of N cannot affect the domains of the X variables.

Let $Y = \{X_{p_1}, \dots, X_{p_k}\}$ be a maximum cardinality subset of variables of X whose ranges are pairwise disjoint (i.e., $\text{range}(X_{p_i}) \cap \text{range}(X_{p_j}) = \emptyset, \forall i, j \in 1..k, i \neq j$). Let $I_Y = \{[b_i, c_i] \mid b_i = \min(X_{p_i}), c_i = \max(X_{p_i}), X_{p_i} \in Y\}$ be the corresponding ordered set of disjoint ranges of the variables in Y . It has been shown in [9] that $|Y| = \text{card}_\downarrow(X)$.

Consider the interval $[b_i, c_i] \in I_Y$. Constraints (5) ensure that the variables $M_{b_i c_i}, i = [1, \dots, k]$ are greater than or equal to 1 and constraints (6) ensure that the variable M_{1d} is greater than or equal to the sum of lower bounds of variables $M_{b_i c_i}, i = [1, \dots, k]$, because intervals $[b_i, c_i]$ are disjoint. Therefore, the variable N is greater than or equal to $\text{card}_\downarrow(X)$ and it is bound consistent.

We show that when N is BC and $\text{dom}(N) \neq \{\text{card}_\downarrow(X)\}$, all X variables are BC. Take any assignment $S \in \prod_{X_i \in X} \text{range}(X_i)$ such that $\text{card}(S) = \text{card}_\downarrow(X)$. Let $S[X_i \leftarrow b]$ be the assignment S where the value of X_i in S has been replaced by b , one of the bounds of X_i . We know that $\text{card}(S[X_i \leftarrow b]) \in [\text{card}(S) - 1, \text{card}(S) + 1] = [\text{card}_\downarrow(X) - 1, \text{card}_\downarrow(X) + 1]$ because only one variable has been flipped. Hence, any assignment (S, p) with $p \geq \text{card}_\downarrow(X) + 1$ is a bound support. $\text{dom}(N)$ necessarily contains such a value p by assumption.

The only case when pruning might occur is if the variable N is ground and $\text{card}_\downarrow(X) = N$. Constraints (6) imply that M_{1d} equals the sum of variables $M_{1, b_1-1} + M_{b_1, c_1} + M_{c_1+1, b_2-1} \dots + M_{b_N, c_N} + M_{c_N+1, d}$. The lower bound of the variable M_{c_i, b_i} is greater than one and there are $|Y| = \text{card}_\downarrow(X) = N$ of these intervals. Therefore, by constraint (7), the upper bound of variables $M_{c_{i-1}+1, b_i-1}$ that correspond to intervals outside the set I_Y are forced to zero.

There are $O(nd^2)$ constraints (4) and constraints (5) that can be woken $O(d)$ times down the branch of the search tree. Each requires $O(1)$ time for a total of $O(nd^3)$ down the branch. There are $O(d^2)$ constraints (6) which can be woken $O(n)$ times down the branch and each invocation takes $O(1)$ time. This gives a total of $O(nd^2)$. The final complexity down the branch of the search tree is therefore $O(nd^3)$. \square

The proof of theorem 3 also provides the corollary that enforcing range on consistency on constraints 4 enforces range consistency on ATMOSTNVALUE. Note that theorem 3 shows that the BC propagator of ATMOSTNVALUE [12] is not algorithmically global with respect to time, as BC can be achieved with a decomposition with comparable time complexity. On the other hand, the $O(nd^2)$ space complexity of this decomposition suggests that it is algorithmically global with respect to space. Of course, we only provide upper bounds here, so it may be that ATMOSTNVALUE is not algorithmically global with respect to either time or space.

5 Faster Decompositions

We can improve how the solver handles this decomposition of the ATMOSTNVALUE constraint by adding implied constraints and by implementing specialized propagators. Our first improvement is to add an implied constraint and enforce BC on it:

$$M_{1d} = \sum_{i=1}^d M_{ii} \quad (8)$$

This does not change the asymptotic complexity of reasoning with the decomposition, nor does it improve the level of propagation achieved. However, we have found that the fixed point of propagation is reached quicker in practice with such an implied constraint.

Our second improvement decreases the asymptotic complexity of enforcing BC on the decomposition of Section 4. The complexity is dominated by reasoning with constraints (4) which channel from X_i to A_{ilu} and thence onto M_{lu} (through constraints (5)). If constraints (4) are not woken uselessly, enforcing BC costs $O(1)$ per constraint down the branch. Unfortunately, existing solvers wake up such constraints as soon as a bound is modified, thus giving a cost in $O(d)$. We therefore implemented a specialized propagator to channel between X_i and M_{lu} efficiently. To be more precise, we remove the $O(nd^2)$ variables A_{ilu} and replace them with $O(nd)$ Boolean variables Z_{ij} . We then add the following constraints

$$Z_{ij} = 1 \iff X_i \leq j \quad 1 \leq j \leq d \quad (9)$$

$$Z_{i(l-1)} = 1 \vee Z_{iu} = 0 \vee M_{lu} > 0 \quad 1 \leq l \leq u \leq d, 1 \leq i \leq n \quad (10)$$

These constraints are enough to channel changes in the bounds of the X variables to M_{lu} . There are $O(nd)$ constraints (9), each of which can be propagated in time $O(d)$ over a branch, for a total of $O(nd^2)$. There are $O(nd^2)$ clausal constraints (10) and each of them can be made BC in time $O(1)$ down a branch of the search tree, for a total cost of $O(nd^2)$. Since channeling dominates the asymptotic complexity of the entire decomposition of Section 4, this improves the complexity of this decomposition to $O(nd^2)$. This is similar to the technique used in [5] to improve the asymptotic complexity of the decomposition of the ALL-DIFFERENT constraint.

Our third improvement is to enforce stronger pruning by observing that when $M_{lu} = 0$, we can remove the interval $[l, u]$ from all variables, regardless of whether this modifies their bounds. This corresponds to enforcing RC on constraints (4). Interestingly, this is sufficient to achieve RC on the ATMOSTNVALUE constraint. Unfortunately, constraints (10) cannot achieve this pruning and using constraints (4) increases the complexity of the decomposition back to $O(nd^3)$. Instead we extend the decomposition with $O(d \log d)$ Boolean variables $B_{il(l+2^k)} \in [0, 1]$, $1 \leq i \leq n$, $1 \leq l \leq d$, $0 \leq k \leq \lfloor \log d \rfloor$. The following constraint ensures that $B_{ijj} = 1 \iff X_i = j$.

$$\text{DOMAINBITMAP}(X_i, [B_{i11}, \dots, B_{idd}]) \quad (11)$$

Clearly we can enforce RC on this constraint in time $O(d)$ over a branch, and $O(nd)$ for all variables X_i . We can then use the following clausal constraints to channel from variables M_{lu} to these variables and on to the X variables. These constraints are posted for every $1 \leq i \leq n$, $1 \leq l \leq u \leq d$, $1 \leq j \leq d$ and integers k such that $0 \leq k \leq \lfloor \log d \rfloor$:

$$B_{ij(j+2^{k+1}-1)} = 1 \vee B_{ij(j+2^k-1)} = 0 \quad (12)$$

$$B_{ij(j+2^{k+1}-1)} = 1 \vee B_{i(j+2^k)(j+2^{k+1}-1)} = 0 \quad (13)$$

$$M_{lu} \neq 0 \vee B_{il(l+2^k-1)} = 0 \quad 2^k \leq u - l + 1 < 2^{k+1} \quad (14)$$

$$M_{lu} \neq 0 \vee B_{i(u-2^k+1)u} = 0 \quad 2^k \leq u - l + 1 < 2^{k+1} \quad (15)$$

The variable $B_{il(l+2^k-1)}$, similarly to the variables A_{lu} , is true when $X_i \in [l, l + 2^k - 1]$, but instead of having one such variable for every interval, we only have them for intervals whose length is a power of two. When $M_{lu} = 0$, with $2^k \leq u - l + 1 < 2^{k+1}$, the constraints (I4)–(I5) set to 0 the B variables that correspond to the two intervals of length 2^k that start at l and finish at u , respectively. In turn, the constraints (I2)–(I3) set to 0 the B variables that correspond to intervals of length 2^{k-1} , all the way down to intervals of size 1. These trigger the constraints (I1), so all values in the interval $[l, u]$ are removed from the domains of all variables.

Example. Suppose $X_1 \in [5, 9]$. Then, by (9), $Z_{14} = 0, Z_{19} = 1$ and by (10), $M_{59} > 0$. Conversely, suppose $M_{59} = 0$ and $X_1 \in [1, 10]$. Then, by (I4)–(I5), we get $B_{158} = 0$ and $B_{169} = 0$. From $B_{158} = 0$ and (I2)–(I3) we get $B_{156} = 0, B_{178} = 0, B_{155} = B_{166} = B_{177} = B_{188} = 0$, and by (I1), the interval $[5, 8]$ is pruned from X_1 . Similarly, $B_{169} = 0$ causes the interval $[6, 9]$ to be removed from X_1 , so $X_1 \in [1, 4] \cup \{10\}$.

Note that RC can be enforced on each of these constraints in constant time over a branch. There exist $O(nd \log d)$ of the constraints (I2)–(I3) and $O(nd^2)$ of the constraints (I4)–(I5), so the total time to propagate them all down a branch is $O(nd^2)$.

6 ATLEASTNVALUE Constraint

There is a similar decomposition for the ATLEASTNVALUE constraint. We introduce 0/1 variables, A_{ilu} to represent whether X_i uses a value in the interval $[l, u]$, and integer variables, E_{lu} with domains $[0, n]$ to count the number of times values in $[l, u]$ are re-used, that is, how much the number of variables taking values in $[l, u]$ exceeds the number $u - l + 1$ of values in $[l, u]$. To constrain these introduced variables, we post the following constraints:

$$A_{ilu} = 1 \iff X_i \in [l, u] \quad \forall 1 \leq i \leq n, 1 \leq l \leq u \leq d \quad (16)$$

$$E_{lu} \geq \sum_{i=1}^n A_{ilu} - (u - l + 1) \quad \forall 1 \leq l \leq u \leq d \quad (17)$$

$$E_{1u} = E_{1k} + E_{(k+1)u} \quad \forall 1 \leq k < u \leq d \quad (18)$$

$$N \leq n - E_{1d} \quad (19)$$

Running Example. Consider the decomposition of an ATLEASTNVALUE constraint over the following variables and values:

	1	2	3	4	5
X_1	*	*	*	*	*
X_2		*			*
X_3	*	*	*		
X_4				*	
X_5		*	*		
N	*	*		*	

Bound consistency reasoning on the decomposition will make the following inferences. As $dom(X_i) \subseteq [2, 4]$ for $i \in 2..5$, from (I6) we get $A_{i24} = 1$ for $i \in 2..5$. Hence, by (I7), $E_{24} \geq 1$. By (I8), $E_{15} = E_{14} + E_{55}$, $E_{14} = E_{11} + E_{24}$. Since $E_{24} \geq 1$ we deduce that $E_{15} \geq 1$. Finally, from (I9) and the fact that $n = 5$, we get $N \leq 4$. This gives us bound consistent domains for the ATLEASTNVALUE constraint.

We now prove that this decomposition does not hinder propagation in general.

Theorem 4. *BC on the constraints (16) to (19) is equivalent to BC on ATLEASTNVALUE $([X_1, \dots, X_n], N)$, and takes $O(nd^3)$ time to enforce down the branch of the search tree.*

Proof: First note that changing the domains of the X variables cannot affect the lower bound of N by the ATLEASTNVALUE constraint and, conversely, changing the upper bound of N cannot affect the domains of the X variables.

It is known [12] that $\text{card}_1(X)$ is equal to the size of a maximum matching M in the value graph of the constraint. Since $N \leq n - E_{1,d}$, we show that the lower bound of $E_{1,d}$ is equal to $n - |M|$.¹ We first show that we can construct a matching $M(E)$ of size $n - \min(E_{1,d})$, then show that it is a maximum matching. The proof uses a partition of the interval $[1, d]$ into a set of maximal saturated intervals $I = \{[b_j, c_j]\}$, $j = 1, \dots, k$ such that $\min(E_{b_j, c_j}) = \sum_{i=1}^n \min(A_{ib_j c_j}) - (c_j - b_j + 1)$ and a set of unsaturated intervals $\{[b_j, c_j]\}$ such that $\min(E_{b_j, c_j}) = 0$.

Let $I = \{[b_j, c_j] \mid j \in [1 \dots k]\}$ be the ordered set of maximal intervals such that $\min(E_{b_j, c_j}) = \sum_{i=1}^n \min(A_{ib_j c_j}) - (c_j - b_j + 1)$. Note that the intervals in I are disjoint otherwise intervals are not maximal. An interval $[b_i, c_i]$ is smaller than $[b_j, c_j]$ iff $c_i < b_j$. We denote the union of the first j intervals $D_I^j = \bigcup_{i=1}^j [b_i, c_i]$, $j = [1, \dots, k]$, $p = |D_I^k|$ and the variables whose domain is inside one of intervals I $X_I = \{X_{p_i} \mid \text{dom}(X_{p_i}) \subseteq D_I^k\}$.

Our construction of a matching uses two sets of variables, X_I and $X \setminus X_I$. First, we identify the cardinality of these two sets. Namely, we show that the size of the set X_I is $p + \min(E_{1,d})$ and the size of the set $X \setminus X_I$ is $n - (p + \min(E_{1,d}))$.

Intervals I are saturated therefore each value from these intervals are taken by a variable in X_I . Therefore, X_I has size at least p . Moreover, there exist $\min(E_{1,d})$ additional variables that take values from D_I^k , because values from intervals between two consecutive intervals in I do not contribute to the lower bound of the variable E by construction of I . Therefore, the number of variables in D_I^k is at least $p + \min(E_{1,d})$. Note that constraints (18) imply that $E_{1,d}$ equals the sum of variables $E_{1, b_1-1} + E_{b_1, c_1} + E_{c_1+1, b_2-1} \dots + E_{b_k, c_k} + E_{c_k+1, d}$. As intervals in I are disjoint then $\sum_{i=1}^k \min(E_{b_i, c_i}) = |X_I| - p$. If $|X_I| > p + \min(E_{1,d})$ then $\sum_{i=1}^k \min(E_{b_i, c_i}) > \min(E_{1,d})$ and the lower bound of the variable $E_{1,d}$ will be increased. Hence, $|X_I| = p + \min(E_{1,d})$.

Since all these intervals are saturated, we can construct a matching M_I of size p using the variables in X_I . The size of $X \setminus X_I$ is $n - p - \min(E_{1,d})$. We show by contradiction that we can construct a matching $M_{D - D_I^k}$ of size $n - p - \min(E_{1,d})$ using the variables in $X \setminus X_I$ and the values $D - D_I^k$.

Suppose such a matching does not exist. Then, there exists an interval $[b, c]$ such that $|(D \setminus D_I^k) \cap [b, c]| < \sum_{i \in X \setminus X_I} \min(A_{ibc})$, i.e., after consuming the values in I with variables in X_I , we are left with fewer values in $[b, c]$ than variables whose domain is contained in $[b, c]$. We denote $p' = |[b, c] \cap D_I^k|$, so that p' is the number of values inside the interval $[b, c]$ that are taken by variables in X_I . The total number of

¹ We assume that $E_{1,d}$ is not pruned by other constraints.

variables inside the interval $[b, c]$ is greater than or equal to $\sum_{i=1}^n \min(A_{ibc})$. The total number of variables X_I inside the interval $[b, c]$ equals to $p' + \min(E_{b,c})$. Therefore, $\sum_{i \in X \setminus X_I} \min(A_{ibc}) \leq \sum_{i=1}^n \min(A_{ibc}) - p' - \min(E_{b,c})$. On the other hand, the number of values that are not taken by the variables X_I in the interval $[b, c]$ is $c - b + 1 - p'$. Therefore, we obtain the inequality $c - b + 1 - p' < \sum_{i=1}^n \min(A_{ibc}) - p' - \min(E_{b,c})$ or $\min(E_{bc}) < \sum_{i=1}^n \min(A_{ibc}) - (c - b + 1)$. By construction of I , $\sum_{i=1}^n \min(A_{ibc}) - (c - b + 1) < \min(E_{bc})$, otherwise the intervals in I that are subsets of $[b, c]$ are not maximal. This leads to a contradiction, so we can construct a matching $M(E)$ of size $n - \min(E_{1d})$.

Now suppose that $M(E)$ is not a maximum matching. This means that $\min(E_{1d})$ is overestimated by propagation on (16) and (19). Since $M(E)$ is not a maximum matching, there exists an augmenting path of $M(E)$, that produces M' , such that $|M'| = |M(E)| + 1$. This new matching covers all the values that $M(E)$ covers and one additional value q . We show that q cannot belong to the interval $[1, d]$.

The value q cannot be in any interval in I , because all values in $[b_i, c_i] \in I$ are used by variables whose domain is contained in $[b_i, c_i]$. In addition, q cannot be in an interval $[b, c]$ between two consecutive intervals in I , because those intervals do not contribute to the lower bound of E_{1d} . Thus, M' cannot cover more values than $M(E)$ and they must have the same size, a contradiction.

We show that when N is BC and $\text{dom}(N) \neq \{\text{card}_\uparrow(X)\}$, all X variables are BC. Take any assignment $S \in \Pi_{X_i \in X} \text{range}(X_i)$ such that $\text{card}(S) = \text{card}_\uparrow(X)$. Let $S[X_i \leftarrow b]$ be the assignment S where the value of X_i in S has been replaced by b , one of the bounds of X_i . We know that $\text{card}(S[X_i \leftarrow b]) \in [\text{card}(S) - 1, \text{card}(S) + 1] = [\text{card}_\uparrow(X) - 1, \text{card}_\uparrow(X) + 1]$ because only one variable has been flipped. Hence, any assignment (S, p) with $p \leq \text{card}_\uparrow(X) - 1$ is a bound support. $\text{dom}(N)$ necessarily contains such a value p by assumption.

We now show that if $N = \text{card}_\uparrow(X)$, enforcing BC on the constraints (16)–(19) makes the variables X BC with respect to the ATLEASTNVALUE constraint. We first observe that in a bound support, variables X must take the maximum number of different values because $N = \text{card}_\uparrow(X)$. Hence, in a bound support, variables X that are not included in a saturated interval will take values outside any saturated interval they overlap and they all take different values. We recall that $\min(E_{1d}) = n - |M| = n - \text{card}_\uparrow(X)$. Hence, by constraint (19), $E_{1d} = n - N$. We recall the size of set X_I equals $p + E_{1d}$. Constraints (18) imply that E_{1d} equals the sum of variables $E_{1,b_1-1} + E_{b_1,c_1} + E_{c_1+1,b_2-1} \dots + E_{b_k,c_k} + E_{c_k+1,d}$ and $\sum_{i=1}^k \min(E_{b_i,c_i}) = |X_I| - p = \min(E_{1d}) = \max(E_{1d})$. Hence, by constraints (18), the upper bounds of all variables E_{b_i,c_i} that correspond to the saturated intervals are forced to $\min(E_{b_i,c_i})$. Thus, by constraints (16) and (17), all variables in $X \setminus X_I$ have their bounds pruned if they belong to D_I^k . By constraints (18) again, the upper bounds of all variables E_{lu} that correspond to the unsaturated intervals are forced to take value 0, and all variables $E_{l'u'}$ with $[l', u'] \subseteq [l, u]$ are forced to 0 as well. Thus, by constraints (16) and (17), all variables in $X \setminus X_I$ have their bounds pruned if they belong to a Hall interval of other variables in $X \setminus X_I$. This is what BC on the ALL-DIFFERENT constraint does [5].

There are $O(nd^2)$ constraints (16) that can be woken $O(d)$ times down the branch of the search tree in $O(1)$, so a total of $O(nd^3)$ down the branch. There are $O(d^2)$

constraints (17) which can be propagated in time $O(n)$ down the branch for a $O(nd^2)$. There are $O(d^2)$ constraints (18) which can be woken $O(n)$ times each down the branch for a total cost in $O(n)$ time down the branch. Thus a total of $O(nd^2)$. The final complexity down the branch of the search tree is therefore $O(nd^3)$. \square

The complexity of enforcing BC on ATLEASTNVALUE can be improved to $O(nd^2)$ in a way similar to that described in Section 5 and in [5]. As with ATMOSTNVALUE, enforcing RC on constraints (16) enforces RC on ATLEASTNVALUE, but in this case we cannot reduce the complexity below $O(nd^3)$. Similarly to ATMOSTNVALUE, theorem 4 shows that the bound consistency propagator of ATLEASTNVALUE is not algorithmically global with respect to time and provides evidence that it is algorithmically global with respect to space.

7 Experimental Results

As noted before, the main value of these decompositions is theoretical: demonstrating that the bound consistency propagator of [12] for the NVALUE constraint can be simulated using a simple decomposition with comparable time complexity over a branch of the search tree but greater space complexity. To see when this space complexity hits, we performed some experiments. We used a benchmark problem, the dominating set of the Queen’s graph used in previous studies of NVALUE [13] and ran experiments with Ilog Solver 6.2 and Ilog CPLEX 9.1 on an Intel Xeon 4 CPU, 2.0 Ghz, 4Gb RAM. The dominating set of the Queen’s graph problem is to put the minimum number of queens on a $n \times n$ chessboard, so that each square either contains a queen or is attacked by one. This is equivalent to the dominating set problem of the Queen’s graph. Each vertex in the Queen’s graph corresponds to a square of the chessboard and there exists an edge between two vertices iff a queen from one square can attack a queen from the other square. To model the problem, we use a variable X_i for each square, and values from 1 to n^2 and post a single ATMOSTNVALUE($[X_1, \dots, X_{n^2}], N$) constraint. The value j belongs to $dom(X_i)$ iff there exists an edge (i, j) in the Queen’s graph or $j = i$. For $n \leq 120$, all minimum dominating sets for the Queen’s problem are either of size $\lceil n/2 \rceil$ or $\lceil n/2 + 1 \rceil$ [15]. We therefore only solved instances for these two values of N .

We compare our decomposition with the simple decomposition of the ATMOSTNVALUE constraint in Ilog Solver and Ilog CPLEX solvers. The simple decomposition is the one described in Section 3.1 except that in constraint (3), we replace “=” by “ \leq ”. We denote this decomposition $Occs$ and $Occs^{CPLEX}$ in Ilog Solver and CPLEX, respectively. To encode this decomposition into an integer linear program, we introduce literals b_{ij} , $i, j \in [1, n^2]$ and use a direct encoding with b_{ij} for the truth of $X_i = j$ and channeling inequalities $1 - b_{ij} + B_j \geq 1$, $i, j \in [1, n^2]$. We use the direct encoding of variables domains to avoid using logic constraints, like disjunction and implication constraints in CPLEX. The default transformation of logic constraints in CPLEX appears to generate large ILP models and this slows down the search.

The BC decomposition is described in Section 4, which we call $Pyramid_{BC}$ and $Pyramid_{BC}^{CPLEX}$ in Ilog Solver and CPLEX, respectively. In Ilog Solver, as explained in Section 5, we channel the variables X_i directly to the pyramid variables M_{lu} to avoid introducing many auxiliary variables A_{ilu} and we add the redundant constraint

Table 1. Backtracks and runtime (in seconds) to solve the dominating set problem for the Queen’s graph

n	N	<i>Occs</i>		<i>Pyramid_{BC}</i>		<i>Occs^{CPLEX}</i>		<i>Pyramid_{BC}^{CPLEX}</i>	
		backtracks	time	backtracks	time	backtracks	time	backtracks	time
5	3	34	0.01	7	0.00	1	0.05	3	0.4
6	3	540	0.16	118	0.03	2	0.16	183	9.6
7	4	195,212	84.50	83,731	15.49	130,010	1802.49	63	15.8
8	5	390,717	255.64	256,582	58.42	24,588	585.07	30	41.28

$\sum_{i=1}^{n^2} M_{ii} = M_{1,n^2}$ to the decomposition to speed up the propagation across the pyramid. We re-implemented the ternary sum constraint in Ilog for a 30% speedup.

To encode the BC decomposition into an integer linear program, we use the linear encoding of variables domains [16]. We introduce literals c_{ij} for the truth of $X_i \leq j$, and the channeling inequalities of the form $c_{i(l-1)} + 1 - c_{iu} + M_{lu} \geq 1$. We again add the redundant constraint $\sum_{i=1}^{n^2} M_{ii} = M_{1,n^2}$. Finally, we post constraints (6) as lazy constraints in CPLEX. Lazy constraints are constraints that are not expected to be violated when they are omitted. These constraints are not taken into account in the relaxation of the problem and are only included when they violate an integral solution.

Results of our experiments are presented in Table 1. Our BC decomposition performs better than the *Occs* decomposition, both in runtime and in number of backtracks needed by Ilog Solver or CPLEX. CPLEX is slower per node than Ilog Solver. However, CPLEX usually requires fewer backtracks compared to ILOG Solver. Interestingly CPLEX performs well with the BC decomposition. The time to explore each node is large, reflecting the size of decomposition, but the number of search nodes explored is small. We conjecture that integer linear programming methods like CPLEX will perform in a similar way with other decompositions of global constraints which do not hinder propagation (e.g. the decompositions we have proposed for ALL-DIFFERENT and GCC). Finally, the best results here are comparable with those for the ATMOSTNVALUE bounds consistency propagator in [13].

8 Other Related Work

Bessiere *et al.* consider a number of different methods to compute a lower bound on the number of values used by a set of variables [13]. One method is based on a simple linear relaxation of the minimum hitting set problem. This gives a propagation algorithm that achieves a level of consistency strictly stronger than bound consistency on the NVALUE constraint. Cheaper approximations are also proposed based on greedy heuristics and an approximation for the independence number of the interval graph due to Turán. Decompositions have been given for a number of other global constraints. For example, Beldiceanu *et al.* identify conditions under which global constraints specified as automata can be decomposed into signature and transition constraints without hindering propagation [17]. As a second example, many global constraints can be decomposed using ROOTS and RANGE which can themselves be propagated effectively using simple decompositions [18]. As a third example, the REGULAR and CFG constraints can be

decomposed without hindering propagation [19,20]. As a fourth example, decompositions of the SEQUENCE constraint have been shown to be effective [21]. Most recently, we demonstrated that the ALL-DIFFERENT and GCC constraint can be decomposed into simple primitive constraints without hindering bound consistency propagation [5]. These decompositions also introduced variables to count variables using values in an interval. For example, the decomposition of ALL-DIFFERENT ensures that no interval has more variables taking values in the interval than the number of values in the interval. Using a circuit complexity lower bound, we also proved that there is no polynomial sized SAT decomposition of the ALL-DIFFERENT constraint (and therefore of its generalizations like NVALUE) on which unit propagation achieves domain consistency [6]. Our use of “pyramid” variables is similar to the use of the “partial sums” variables in the encoding of the SEQUENCE constraint in [21]. This is related to the cumulative sums computed in [22].

9 Conclusions

We have studied a number of decompositions of the NVALUE constraint. We have shown that a simple decomposition can simulate the bound consistency propagator for NVALUE [12] with comparable time complexity but with a much greater space complexity. This supports the conclusion that the benefit of a global propagator may often not be in saving time but in saving space. Our other theoretical contribution is to show the first range consistency algorithm for NVALUE, that runs in $O(nd^3)$ time and $O(nd^2)$ space. These results are largely interesting from a theoretical perspective. They help us understand the globality of global constraints. They highlight that saving space may be one of the important advantages provided by propagators for global constraints. We have seen that the space complexity of decompositions of many propagators equals the worst case time complexity (e.g. for the ALL-DIFFERENT, GCC, AMONG, LEX, REGULAR, CFG and SEQUENCE constraints). For global constraints like REGULAR, the space complexity of the decompositions does not appear to be that problematic. However, for global constraints like NVALUE, the space complexity of the decompositions is onerous. This space complexity seems hard to avoid. For example, consider encodings into satisfiability and unit propagation as our inference method. As unit propagation is linear in time in the size of the encoding, it is somewhat inevitable that the size of any encoding is the same as the worst-case time complexity of any propagator that is being simulated. One other benefit of these decompositions is that they help us explore the interface between constraint and integer linear programming. For example, we saw that an integer programming solver performed relatively well with these decompositions.

Acknowledgements. NICTA is funded by the Department of Broadband, Communications and the Digital Economy, and the ARC. Christian Bessiere is supported by ANR project ANR-06-BLAN-0383-02, and George Katsirelos by ANR UNLOC project: ANR 08-BLAN-0289-01. We thank Lanbo Zheng for experimental help.

References

1. Régim, J.C.: A filtering algorithm for constraints of difference in CSPs. In: Proc. of the 12th National Conf. on AI, pp. 362–367. AAAI, Menlo Park (1994)
2. Leconte, M.: A bounds-based reduction scheme for constraints of difference. In: Proc. of 2nd Int. Workshop on Constraint-Based Reasoning, Constraint 1996 (1996)
3. Puget, J.: A fast algorithm for the bound consistency of alldiff constraints. In: 15th National Conf. on Artificial Intelligence, pp. 359–366. AAAI, Menlo Park (1998)
4. Bessiere, C., Hentenryck, P.V.: To be or not to be... a global constraint. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 789–794. Springer, Heidelberg (2003)
5. Bessiere, C., Katsirelos, G., Narodytska, N., Quimper, C.G., Walsh, T.: Decompositions of all different, global cardinality and related constraints. In: Proc. of 21st IJCAI, pp. 419–424 (2009)
6. Bessiere, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit complexity and decompositions of global constraints. In: Proc. of 21st IJCAI, pp. 412–418 (2009)
7. Bessiere, C., Hebrard, E., Hnich, B., Walsh, T.: The complexity of global constraints. In: Proc. of the 19th National Conf. on AI. AAAI, Menlo Park (2004)
8. Pacht, F., Roy, P.: Automatic generation of music programs. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 331–345. Springer, Heidelberg (1999)
9. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the NVALUE constraint. In: Barták, R., Milano, M. (eds.) CPAIOR 2005. LNCS, vol. 3524, pp. 79–93. Springer, Heidelberg (2005)
10. Bessiere, C., Hebrard, E., Hnich, B., Walsh, T.: The complexity of global constraints. *Constraints* 12, 239–259 (2007)
11. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C.G., Walsh, T.: The parameterized complexity of global constraints. In: Proc. of the 23rd National Conf. on AI, pp. 235–240. AAAI, Menlo Park (2008)
12. Beldiceanu, N.: Pruning for the minimum constraint family and for the number of distinct values constraint family. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 211–224. Springer, Heidelberg (2001)
13. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the NVALUE constraint. *Constraints* 11, 271–293 (2006)
14. Stergiou, K., Walsh, T.: The difference all-difference makes. In: Proc. of 16th IJCAI (1999)
15. Östergård, P., Weakley, W.: Values of domination numbers of the queen’s graph. *The Electronic Journal of Combinatorics* 8 (2001)
16. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation = lazy clause generation. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 544–558. Springer, Heidelberg (2007)
17. Beldiceanu, N., Carlsson, M., Debruyne, R., Petit, T.: Reformulation of Global Constraints Based on Constraints Checkers. *Constraints* 10, 339–362 (2005)
18. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: The range and roots constraints: Specifying counting and occurrence problems. In: Proc. of 19th IJCAI, pp. 60–65 (2005)
19. Quimper, C.G., Walsh, T.: Global grammar constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 751–755. Springer, Heidelberg (2006)
20. Quimper, C.G., Walsh, T.: Decomposing global grammar constraints. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 590–604. Springer, Heidelberg (2007)
21. Brand, S., Narodytska, N., Quimper, C.G., Stuckey, P., Walsh, T.: Encodings of the Sequence Constraint. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 210–224. Springer, Heidelberg (2007)
22. van Hoes, W.J., Pesant, G., Rousseau, L.M., Sabharwal, A.: Revisiting the Sequence Constraint. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 620–634. Springer, Heidelberg (2006)

Propagating the Bin Packing Constraint Using Linear Programming*

Hadrien Cambazard and Barry O’Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.cambazard,b.osullivan}@4c.ucc.ie

Abstract. The state-of-the-art global constraint for bin packing is due to Shaw. We compare two linear continuous relaxations of the bin packing problem, based on the DP-flow and Arc-flow models, with the filtering of the bin packing constraint. Our experiments show that we often obtain significant improvements in runtime. The DP-flow model is a novel formulation of the problem.

1 Introduction

The one-dimensional bin packing problem is ubiquitous in operations research. It is typically defined as follows. Given a set $S = \{s_1, \dots, s_n\}$ of n indivisible items each of a known positive size s_i , and m bins each of capacity C , can we pack all n items into the m bins such that the sum of sizes of the items in each bin does not exceed C ? The one-dimensional bin packing problem is NP-Complete. Amongst the many applications of this problem are timetabling, scheduling, stock cutting, television commercial break scheduling, and container packing.

Our *motivation* comes from a real-world timetabling problem in the Dental School at University College Cork. An interesting characteristic of this problem is that the core challenge relates to solving one-dimensional bin packing problems. This contrasts with many other school timetabling problems which often have challenging list colourings at their core. Our *objective* is to compare two continuous relaxations of the *bin packing* problem with the state-of-the-art filtering algorithm used in the constraint programming community [9]. Continuous relaxations have been developed for many global constraints including *cumulative* [6], *all-different*, *element* and others [5]. We believe that continuous relaxations can be successfully used for the *bin packing* and *knapsack* constraints [9,10]. This paper presents our initial results in this direction.

2 Linear Programming Formulations for Bin Packing

Numerous linear programming models have been proposed for the bin packing problem [3]. A standard linear model is the following. For each bin $j \in \{1, \dots, m\}$ we introduce a binary variable y_j which we set to 1 if bin j is used in the packing, and 0

* This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886).

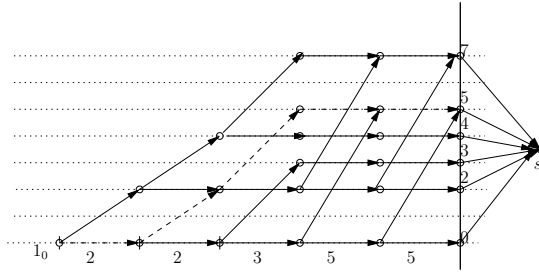


Fig. 1. An example of the graph used by the DP-FLOW model on $S = \{2,2,3,5,5\}$, $C = 7$

otherwise. For each item $i \in \{1, \dots, n\}$ and each bin j we introduce a binary variable x_{ij} which we set to 1 if item i is packed into bin j , and 0 otherwise. The full model is:

$$\begin{aligned}
 & \text{minimize} && \sum_{j=1}^m y_j \\
 & \sum_{j=1}^m x_{ij} = 1, && \forall i \in \{1, \dots, n\} \\
 & \sum_{i=1}^n s_i \times x_{ij} \leq C y_j, && \forall j \in \{1, \dots, m\} \\
 & x_{ij} \in \{0, 1\}, y_i \in \{0, 1\}
 \end{aligned} \tag{1}$$

The lower bound on the number of bins obtained by solving the continuous relaxation of this model is referred to as L_1 in the literature. It has been shown to be equal to $\lceil \sum_{i=1}^n s_i / C \rceil$. Many other lower bounds have been designed, amongst which the most widely used is the one due to Martello and Toth [7], referred to as L_2 . In the remainder of this section we will present two alternative formulations that give better lower bounds on the number of bins. The first one, referred to the DP-FLOW model, is novel. The second one is known as the ARC-FLOW model [2].

The DP-FLOW Model. We consider the directed graph construction used by Trick [10] to build a propagator for the knapsack constraint using dynamic programming; an example is presented in Figure 1. Consider a layered graph $G(V, A)$ with $n + 1$ layers labelled from 1 to $n + 1$, and a sink node, s . The nodes are labelled i_b where i denotes the layer and b a value between 0 and C . A path from the node of the first layer to a node of the last layer represents a packing, i.e. a set of items assigned to the same bin. More specifically a path using an edge starting at layer i between two nodes i_b and $(i + 1)_{b+s_i}$ represents a packing that includes item i , whereas the use of the edge $(i_b, (i + 1)_b)$ excludes item i from the packing. Edges are added between the nodes of the last layer of the graph and the sink node, s . An example of such a graph is shown Figure 1 for the instance $S = \{2, 2, 3, 5, 5\}$, $C = 7$. The packing 2, 3 is the path shown in dashed line.

A solution to the bin packing problem corresponds to a minimum flow problem in this graph, with an additional constraint stating that exactly one oblique edge from each layer must be used. We consider a variable $x_{k,l}$ per edge $(k, l) \in A$, 1_0 being the node of the first layer, and the number of bins is represented by $x_{s,1_0}$, i.e. the flow circulating in the graph:

minimise $x_{s,10}$

$$\sum_{(j,i_b) \in A} x_{j,i_b} - \sum_{(i_b,k) \in A} x_{i_b,k} = \begin{cases} -x_{s,10} & \text{if } i_b = 1_0, \\ 0 & \text{if } i_b \text{ s.t } i \in [2, n + 1], b \in [0, C] \\ x_{s,10} & \text{if } i_b = s \end{cases}$$

$$\sum_{(i_b,(i+1)_{b+s_i}) \in A} x_{(i_b,(i+1)_{b+s_i})} = 1 \quad i \in [1, n]$$

$$x_{k,l} \geq 0, \text{ integer} \quad \forall (k, l) \in A$$

A solution to the bin packing problem can be obtained by decomposing the resulting flow into paths connecting the source node to the nodes of the final layer. This flow decomposition is possible because the graph is acyclic. The number of variables and constraints in this model is $\mathcal{O}(nC)$ since each node in the graph has at most two outgoing edges. Notice that the formulation depends on the ordering of the items used to order the layers of the graph; the size of the graph as well as the strength of the formulation are also affected by this ordering.

The ARC-FLOW Model. Carvalho introduced an elegant ARC-FLOW model for the bin packing problem [2][3]. His model, which we present below, makes explicit the capacity of the bins, and its size depends on the number of items of different sizes rather than the total number of items.

Consider a graph $G(V, A)$ with $C + 1$ nodes labelled from 0 to C in which there is an edge $(i, j) \in A$ if there exists an item of size $j - i$ in S . Additional edges $(i, i + 1)$ are added between consecutive pair of nodes. An example of such a graph is shown in Figure 2 for $S = \{2, 2, 3\}$ with $C = 5$. Any path in this graph corresponds to a packing of a single bin. For example, the path shown in dotted lines in Figure 2 corresponds to a packing of two items of size 2, leaving the remaining capacity of the bin unused (the last edge is a loss edge). More formally, a packing for a single bin corresponds to a flow of one unit between vertices 0 and C . A larger flow corresponds to the same packing into multiple bins.

Such a formulation has many symmetries since the same solution can be encoded by many different flows. Some reduction rules were given by Carvalho that help reduce such symmetries [2]. The graph presented in Figure 3 is a simplified graph for the same example as the one used for the DP-FLOW model. Firstly one can notice that the packings are ordered by decreasing value of the sizes of the items. Secondly, the loss edges before Node 2, which is the smallest item size, have been removed as well. Finally, the number of consecutive edges of a given size is bounded by the number of items of this size. This is why no edges of size 2 are outgoing from Node 4 as this would not correspond to any valid packing. However, all the symetries have not been eliminated.

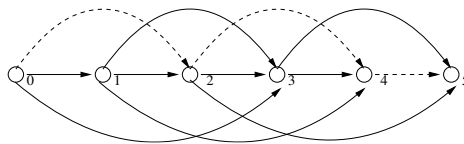


Fig. 2. An example of the graph underlying the ARC-FLOW model for $S = \{2, 2, 3\}$ and $C = 5$. The packing 2, 2 is shown with dotted lines.

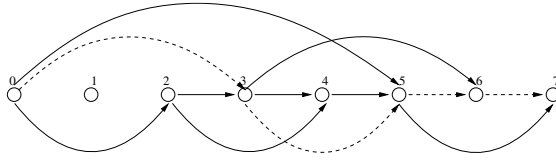


Fig. 3. An example of the graph underlying the ARC-FLOW model for $S = \{2,2,3,5,5\}$, $C = 7$

The bin packing problem can be formulated as a minimum flow between vertex 0 and vertex C with constraints enforcing that the number of edges of a given length used by the flow must be greater than or equal to the number of items of the corresponding size. Variables x_{ij} are associated with the edges (i, j) . x_{C0} denotes the flow variable corresponding to the number of bins used. We will denote by $S' = \{s'_1, \dots, s'_{n'}\}$ the set of different item sizes, and b_i the number of items of size s'_i . n' is the number of items of different sizes. The model is as follows:

$$\begin{aligned}
 & \text{minimise} && x_{C0} \\
 & \sum_{(i,j) \in A} x_{ij} - \sum_{(j,k) \in A} x_{jk} = \begin{cases} -x_{C0} & \text{if } j = 0, \\ 0 & \text{if } j = 1, 2, \dots, C - 1, \\ x_{C0} & \text{if } j = C \end{cases} \\
 & \sum_{(k,k+s'_i) \in A} x_{k,k+s'_i} \geq b_i && i = 1, 2, \dots, n' \\
 & x_{i,j} \geq 0, \text{ integer} && \forall (i, j) \in A
 \end{aligned}$$

A solution can be obtained again by decomposing the flow. The number of variables in this model is $\mathcal{O}(n' C)$. However there are only C flow conservation constraints, as opposed to the nC of our model, which makes this formulation clearly more compact.

3 Dealing with Partial Assignments

The bounds we have presented can be applied during search by transforming a partial assignment into a reduced bin packing problem. Once items are assigned to bins we have a bin packing problem in which some item sizes are missing, since they have already been assigned, and not all bins have the same remaining capacity. Both previous formulations can be modified to handle these cases.

For the DP-FLOW model we can simply add capacities to the edges between the last layer of the graph and the sink node. These capacities express the number of bins that have enough capacity to accommodate the corresponding size. For example, if we dispose of three bins of capacity 10 and an item of size 2 has been assigned to each of the first two bins, then sizes of value 9 and 10 are given a capacity of 1. Additionally, the flows in the oblique edges corresponding to taking items already assigned are enforced to zero. For the ARC-FLOW model a back edge, adding to the overall flow to minimize, can be added from each node corresponding to an available capacity in the reduced bin packing problem. In the previous example an edge would leave Node 8 as well as 10 to go back to Node 0. The b_i values of the linear model are also updated accordingly to reflect the remaining items available.

Table 1. Comparing the quality and time of various lower bounds on the B1 benchmark

	L_1	L_2	DP-FLOW	ARC-FLOW	ARC-FLOW+red
sum	74650.49	77945.76	78114.48	78099.66	78113.4
avg time (in s)	0	0	2.96	0.07	0.02

It is possible to propagate with the LP lower bounds using a similar approach to that adopted by Shaw [9]; that is, we simply commit items to bins, compute the corresponding reduced problem and check whether the bound raises a contradiction or not, in which case the item can be pruned from the corresponding bin. However, that approach is suitable for very fast filtering rules only, but otherwise leads to significant overheads.

4 Experimental Results

We conducted a series of experiments on a single thread on a Dual Quad Core Xeon CPU, 2.66GHz with 12MB of L2 cache per processor and 16GB of RAM overall, running Linux 2.6.25 x64. We put a 3GB limit on memory. CPLEX 12 was used for all the linear models. We used two sets of publicly available instances as benchmarks. The Falkenauer benchmark [4] comprises two classes of instances U and T with four sets of 20 instances in each class containing 120 to 1000 items. Class U comprises item sizes uniformly distributed in $[20, 100]$ to be packed into bins of size 150. Class T consists of triplets of items from $[25, 50]$ to be packed into bins of size 100. Four sets are in class T and instances were generated so that there is no slack. The second benchmark suite used are the B1 and B2 sets studied in [8], made of 720 and 480 instances, respectively. The number of items in these sets vary from 50 to 500; the capacity can reach 1000 in the $B2$ set. We used the first 350 instances of B2. When quoting a benchmark we will use either U, T or B as prefixes followed by the number of items to be packed, unless it is otherwise obvious.

Experiment 1: Comparison of the Lower Bounds on the Number of Bins. We compared four lower bounds: (a) L_1 is the continuous lower bound, (b) L_2 is the well known

Table 2. Comparison between different variants of the bin packing constraint

benchmark		u120	u250	u500	u1000	t60	t120	t249	t501	B1	B2	
CP Sh	Backt	Med	0.00	3.13M	8.68M	2.95M	147.00	5518.00	1.79M	2.41M	0.00	0.00
		Avg	1.41M	4.83M	9.10M	3.01M	216.15	1.54M	2.81M	2.37M	1.15M	3.21M
	Time(s)	Med	0.15	3600.00	3600.07	3600.09	0.07	3.06	3600.07	3600.09	0.34	0.06
		Avg	371.61	1981.41	2992.87	3600.10	0.13	741.78	3265.10	3600.09	497.11	726.09
		StDev	1038.75	1835.99	1302.88	0.06	0.14	1467.85	1033.12	0.06	1224.44	1439.92
NS	19	9	4	0	20	16	2	0	625	280		
CP AC	Backt	Med	15.00	76.00	175.00	285.00	313.00	1093.00	3604.00	8535.50	0.00	0.00
		Avg	25.75	87.80	177.85	268.45	318.20	1100.80	14676.05	16240.85	15.76	132.40
	Time(s)	Med	0.29	0.71	0.96	1.45	13.18	58.10	142.51	208.05	0.21	29.26
		Avg	0.34	0.73	1.02	1.45	13.34	64.92	379.02	408.70	0.54	246.49
		StDev	0.12	0.11	0.15	0.08	4.37	37.95	788.56	757.92	0.96	631.00
NS	20	20	20	20	20	20	19	19	720	343		
CP Sh+AC	Backt	Med	0.00	7.50	21.00	40.50	7.00	33.00	173.50	645.00	0.00	0.00
		Avg	3.95	23.10	24.15	46.10	7.50	39.65	4420.00	3981.50	3.65	68.29
	Time(s)	Med	0.40	1.38	4.44	48.86	3.73	5.65	29.10	76.75	0.48	19.56
		Avg	0.39	1.53	4.86	61.84	3.72	17.43	221.79	272.69	2.28	150.39
		StDev	0.07	0.40	0.93	25.06	1.98	15.10	796.55	785.83	3.92	421.52
NS	20	20	20	20	20	20	19	19	720	349		

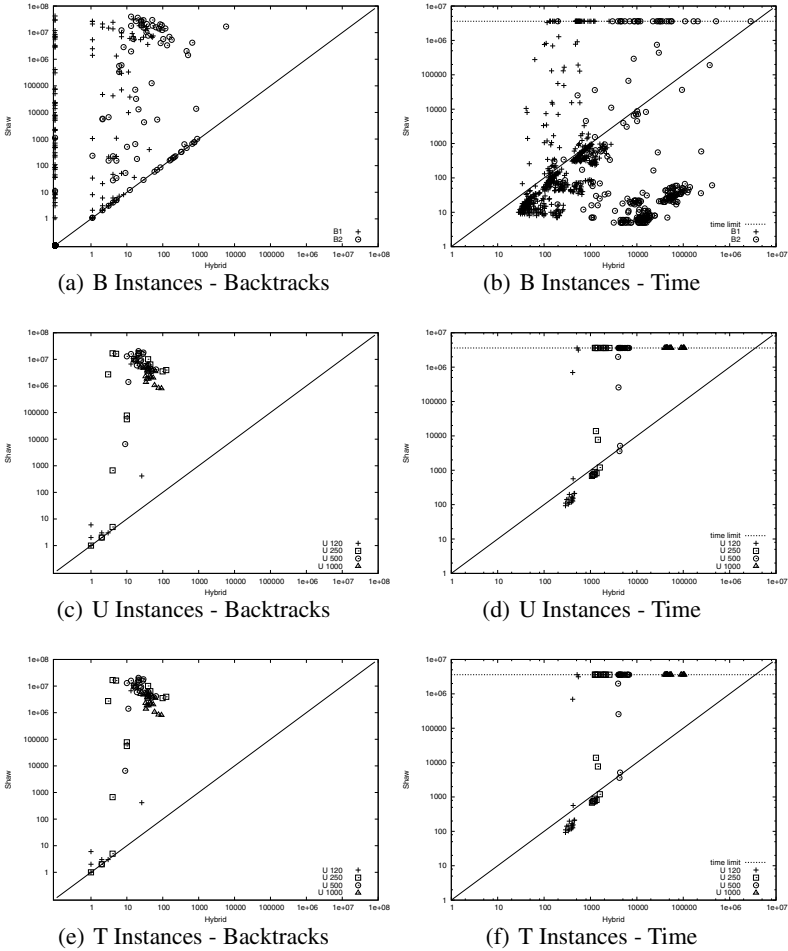


Fig. 4. A detailed comparison of our approach (x-axis) against the bin packing constraint of Shaw on the y-axis - time and backtracks are presented

bound due to Martello and Toth [7], (c) the linear relaxation of the DP-FLOW model where the ordering of the layers in the graph is done by non-decreasing item size, (d) the linear relaxation of the ARC-FLOW model without the simplifications of the graph, and (e) with the reductions. The lower bounds obtained on the Falkenauer benchmark at the root node are not interesting, and mostly equal to the continuous bound L_1 with few exceptions. The second benchmark, B1, exhibits more variety and we report in Table 1 the sum of the lower bounds found on all the instances of the B1 set by the five lower bounds. The linear relaxation of DP-FLOW is the strongest but does not improve ARC-FLOW+red significantly and is also significantly more expensive to compute.

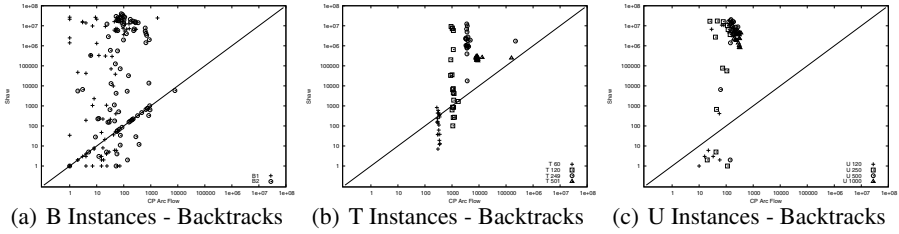


Fig. 5. Comparison of the reasoning capabilities of Shaw’s global constraint against ours

Experiment 2: Comparison with the Bin Packing Constraint. We embedded the ARC-FLOW+red bound within a bin packing global constraint to evaluate the strength of the pruning we would get compared to that obtained using the state-of-the-art global constraint designed by Shaw [9]. We compared these three methods by optimally solving the bin packing problem: “CP Shaw” denotes the constraint described in [9]; “CP Arc-flow” refers to a constraint that simply solves the linear relaxation of the ARC-FLOW model with reductions and uses this lower bound to detect contradiction; “CP Shaw+Arc-flow” first applies the filtering of Shaw’s bin packing constraint and then computes the lower bound of the ARC-FLOW model.

We use similar settings to those in [9]. The standard search heuristic *decreasing best fit* is used, packing items in non-increasing size into the first bin with the least possible space sufficient for the item. On backtracking a bin is removed from the domain of an item and two symmetry breaking rules are used: the bin is also removed from the domain of all equivalent items (items of the same size), and all equivalent bins (bins with same load) are also pruned from the domains of these items. Two dominance rules are also added. The first is applied before creating a new choice point: if all the bins are equivalent for any item it is assigned to the first available one. The second ensures that if an item can fit exactly in the remaining space of a bin it is assigned to this bin.

Our algorithms are implemented with Choco and CPLEX. Shaw’s bin packing constraint is available in Choco, which we used as a baseline. It differs from Shaw’s implementation by using several dual feasible functions [11] that subsume the L_2 bound of Martello and Toth. The results presented in [9] are detailed for the T60 and U120 categories where all instances are solved, but $U120_{19}$ requires 15 hours. We observed a similar behavior: $U120_{19}$ was the only instance not solved within the one hour time limit allowed for the Choco implementation of the bin packing constraint. Table 2 reports the comparison giving for each category the median and average number of backtracks, time (in seconds), as well as the standard deviation in time and the number of instances solved to optimality within the time limit (NS). Clearly the approach presented here significantly improves over Shaw’s bin packing constraint. In Figure 4 we present an instance by instance comparison for each benchmark suite using both Shaw’s global constraint and our hybrid approach. Our approach tends to have a less variation in the effort required to solve a bin packing problem. Finally, in Figure 5 we compare the reasoning capabilities of the ARC-FLOW relaxation against that of Shaw’s global constraint. A point below the diagonal means that Shaw has fewer backtracks than the ARC-FLOW relaxation. The ARC-FLOW alone is usually better than Shaw, by capturing

most of the benefits of more sophisticated filtering while achieving orders of magnitude improvements in time.

5 Conclusion

We have presented a novel direction for handling bin packing constraints in constraint programming. Our approach can give significant improvements in running time. The DP-FLOW and ARC-FLOW models remain to be compared theoretically. A deeper study of the DP-FLOW model focusing on the impact of the ordering of the layers and graph reduction criteria is still to be carried out, although the ARC-FLOW model appears to be much more promising. Furthermore we believe that the graph underlying the ARC-FLOW model scales to much larger problems than the one used by the DP-FLOW model. This immediately suggests that we could apply the same idea to knapsack constraints. The resulting formulation would not be able to provide GAC for the knapsack constraint as opposed to [10] but should allow a very strong propagation in practice while scaling to much bigger knapsacks.

References

1. Clautiaux, F., Alves, C., de Carvalho, J.M.V.: A survey of dual-feasible and superadditive functions. *Annals of Operations Research* (2008)
2. de Carvalho, J.M.V.: Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research* 86, 629–659 (1999)
3. de Carvalho, J.M.V.: LP models for bin packing and cutting stock problems. *European Journal of Operational Research* 141(2), 253–273 (2002)
4. Falkenauer, E.: A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics* 2, 5–30 (1996)
5. Hooker, J.N.: *Integrated Methods for Optimization*. International Series in Operations Research & Management Science. Springer, New York (2006)
6. Hooker, J.N., Yan, H.: A relaxation of the cumulative constraint. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 686–690. Springer, Heidelberg (2002)
7. Martello, S., Toth, P.: Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math.* 28(1), 59–70 (1990)
8. Scholl, A., Klein, R., Jürgens, C.: Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research* 24(7), 627–645 (1997)
9. Shaw, P.: A constraint for bin packing. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 648–662. Springer, Heidelberg (2004)
10. Trick, M.A.: A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals OR* 118(1-4), 73–84 (2003)

Sweeping with Continuous Domains

Gilles Chabert and Nicolas Beldiceanu

École des Mines de Nantes LINA CNRS UMR 6241,
4, rue Alfred Kastler 44300 Nantes, France
{gilles.chabert,nicolas.beldiceanu}@mines-nantes.fr

Abstract. The `geost` constraint has been proposed to model and solve discrete placement problems involving multi-dimensional boxes (packing in space and time). The filtering technique is based on a sweeping algorithm that requires the ability for each constraint to compute a forbidden box around a given fixed point and within a surrounding area. Several cases have been studied so far, including integer linear inequalities. Motivated by the placement of objects with curved shapes, this paper shows how to implement this service for continuous constraints with arbitrary mathematical expressions. The approach relies on symbolic processing and defines a new interval arithmetic.

1 Introduction

Sweeping [4] is a generic technique for filtering with constraints, like propagation for instance. This technique has been fruitfully applied in the context of placement problems (rectangle packing, container loading, warehouse management). In particular, the filtering algorithm of the `geost` constraint [5,8,6], today implemented in different systems like `Choco` [1], `JaCop` [10] or `SICStus` [9], is based on a sweeping loop.

Propagation (e.g., `AC3` [11]) requires a *propagator* for each constraint, that is, an operator that removes inconsistent values. Symmetrically, we will see that sweeping requires an *inflater* for each constraint, that is, an operator that returns a set of unfeasible tuples.

Propagators can be built simply by checking consistency for each value. Likewise, inflaters can be built by enumerating values inside the cross product of domains and checking for inconsistency.

However, such brute force methods, apart from being inefficient, are not possible with continuous domains (that are not countable). In previous publications, inflaters have been proposed for several important classes of constraints: rectangle inclusion and non-overlapping [5], linear equations, distance equations [2] as well as constraints derived from business rules [8]. The method in [8] is generic as it addresses a class of first-order formulae with linear constraints by way of predicates. However, all these methods are restricted to discrete domains and only propose ad-hoc solutions for (a few) nonlinear constraints.

The main contribution of this paper is a generic inflater that works for *any* constraint on continuous domains, as long as the constraint has a mathematical

expression made of usual operators ($+$, \times , $-$, $/$) and functions (sqr, sqrt, sin, etc.). The complexity of this inflater is linear in the length of the constraint expression and optimal (i.e., the corresponding box cannot be extended any more in any dimension) if variables have no multiple occurrences.

Hence, thanks to this new algorithm, the applicability of `geost` can be extended to a significantly larger class of placement problems: nonlinear inequalities can indeed model, e.g., the non-overlapping of objects with curved shapes (balls, cylinders, etc.) in two or three dimensions.

The paper is organized as follows. In Section 2, the sweeping algorithm is recalled, but in a slightly revised form that makes it independent of the nature of domains (discrete or continuous). In Section 3, a generic inflater is described.

2 Sweeping

Since this paper is dedicated to continuous domains, we will need notations to represent and handle intervals of reals. Let us state them now.

Notations. Vectors will be represented by bold face letters. Intervals and boxes (cross products of intervals) will be surrounded by brackets, e.g., $[x]$ and $[\mathbf{x}]$. The symbol \mathbb{IR} represents the set of all intervals. If $[x]$ is an interval, \underline{x} and \bar{x} will stand for the lower and upper bound respectively of $[x]$. If $[\mathbf{x}]$ is a box, the i^{th} component of $[\mathbf{x}]$ is an interval $[x]_i$ (or $[x_i]$) and the bounds of $[x]_i$ will be denoted by \underline{x}_i and \bar{x}_i .

2.1 From Propagation to Sweeping

Consider first a classical propagation of constraints. We are comparing here inference methods (not particular algorithms), but to fix idea one may think about AC3. The principle of propagation is the same for continuous domains as for discrete ones (except that only bounds are usually reduced for practical reasons). In the situation depicted in Figure 1(a), each constraint is able, separately, to filter the domain of a variable. This results in “strips” removed from the domain $[\mathbf{x}] = [x]_1 \times [x]_2$. For instance, when c_1 reduces the left bound of x_1 to \tilde{x}_1 , all the tuples in the hatched strip are proven to be inconsistent.

In the situation of Figure 1(b), propagation is blocked because all the bounds (\underline{x}_1 , \bar{x}_1 , \underline{x}_2 and \bar{x}_2) are consistent (have supports) w.r.t. each constraint. However, let us now assume that, by some means, one can build *forbidden boxes* in the cross domain $[x] \times [y]$, according to the following definition:

Definition 1 (Forbidden box). *Given a constraint c over \mathbb{R}^n , a box $[\mathbf{x}] \subset \mathbb{R}^n$ is called forbidden w.r.t. c if no vector in $[\mathbf{x}]$ satisfies c .*

A first forbidden box $[\mathbf{f}]$, w.r.t. c_1 , can be placed in the lower left corner of $[\mathbf{x}]$ (see Figure 1(b)). A second forbidden box $[\mathbf{f}']$ can then be built above the first one, but by considering this time c_2 . By considering again c_1 a third box $[\mathbf{f}'']$ can be placed again on top of $[\mathbf{f}']$. The union of these boxes covers a strip $[\underline{x}_1, \tilde{x}_1] \times [x]_2$

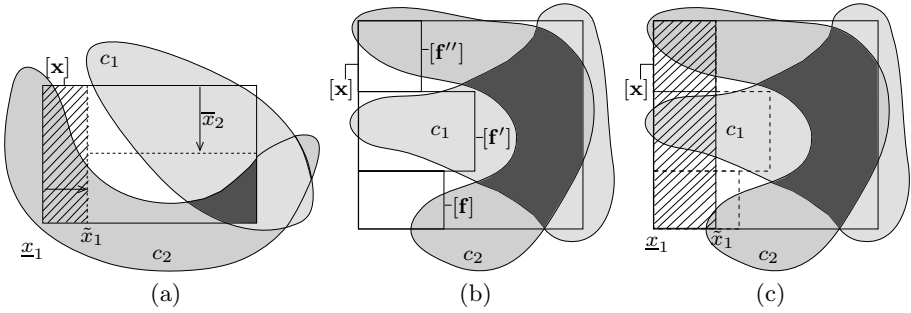


Fig. 1. Comparing propagation and sweeping. The outer rectangle represents the initial box $[x]$. Feasible points of constraints are represented in light gray. The set of solutions in $[x]$ is in dark gray. **(a)**. Propagation can be applied: the left bound of x_1 is reduced by the constraint c_1 (hatched strip). The upper bound of x_2 is then reduced by c_2 . Note that, at this stage, the fixpoint is not reached yet and pruning could be carried on. **(b)**. In this situation, nothing happens with propagation: the bounds of $[x]$ are consistent w.r.t. c_1 and c_2 (taken separately). However a forbidden box can be built w.r.t. each constraint. **(c)** The union of the forbidden boxes allows to prune x_1 .

(see Figure 1(c)) that can be removed, just as in classical propagation. Note that $[\underline{x}_1, \tilde{x}_1]$ is $[f]_1 \cap [f']_1 \cap [f'']_1$, that is, \tilde{x}_1 is the smallest upper bound of the x_1 's, among the forbidden boxes.

There is a duality between propagation and sweeping. Propagation requires each *individual* constraint to provide a *global* filtering (a full strip) while sweeping aggregates the result of *several* constraints, each only providing a *partial* filtering (a forbidden box). The advantage of propagation is clear: a global filtering is an information that can be directly shared by other constraints. But global filtering is clearly difficult to obtain, in general, from a single constraint (especially when related variables have large domains). This is typically observed in placement problems as an empirical fact, even when some objects are initially fixed.

On the contrary, a forbidden box is easier to obtain but does not help the other constraints directly. Computations of forbidden boxes are independent from one constraint to the other. The key idea behind sweeping is to guide these computations so that piling up forbidden boxes maximizes the chance to perform a global filtering quickly. This relies on the concept of *working area*.

2.2 The Working Area

In this section, we shall first consider 2 and then $n > 2$ variables. For the sake of clarity and w.l.o.g., pruning is always assumed to be performed on the left bound of the first variable, x_1 . The other cases are obtained symmetrically, by reordering variables and reversing some inequalities.

A strip like $[\underline{x}_1, \tilde{x}_1] \times [x_2]$ in Figure 1(c) is formed when the union of the projections of forbidden boxes over x_2 entirely cover $[x_2]$. We say, in this case, that x_2 is *saturated*. The property each $x_1 \in [\underline{x}_1, \tilde{x}_1]$ satisfies is then $\forall x_2 \in$

$[x]_2$, (x_1, x_2) is not a solution, which indeed means that $[\underline{x}_1, \tilde{x}_1]$ is inconsistent. Forbidden boxes have to be built within a limited space, bounded below and above by *anchors*:

1. A forbidden box has to be placed on a precise location. For instance, while x_2 is not saturated, the next forbidden box must be placed on the lower left corner above the last one. This location (the lower left corner) defines a *left anchor* for x_1 and x_2 .
2. As said above, \tilde{x}_1 is the lowest upper bound of the x_1 's among the forbidden boxes used to saturate x_2 . This bound can be initialized to \bar{x}_1 and updated incrementally along the process, until saturation of x_2 is achieved. Then, the bound is reset to \bar{x}_1 , for the next strip. This bound defines a *right anchor* for x_1 . The right anchor for x_2 can simply be set to \bar{x}_2 since x_2 is always the first dimension to saturate (in our considered situation).

We call *working area* and denote by $[\mathbf{X}]$ the box delimited by the anchors. It is clear that for a forbidden box $[\mathbf{f}]$, all the part that exceeds the working area is superfluous. Consider for instance Figure 1(b), once $[\mathbf{f}]$ is calculated. The right anchor for x_1 becomes \bar{f}_1 . Then, all the points \mathbf{x} of $[\mathbf{f}']$ with $x_1 > \bar{f}_1$ are useless and, indeed, they won't belong to the final strip. All this extends to n variables thanks to a n -dimensional working area, as shown in Figure 2 for $n = 3$.

For all j , $1 \leq j \leq n$, sweeping over x_j (potentially up to saturation) is recursively based on the saturation of x_{j+1} . When x_{j+1} is saturated, the $(n - j)$ -

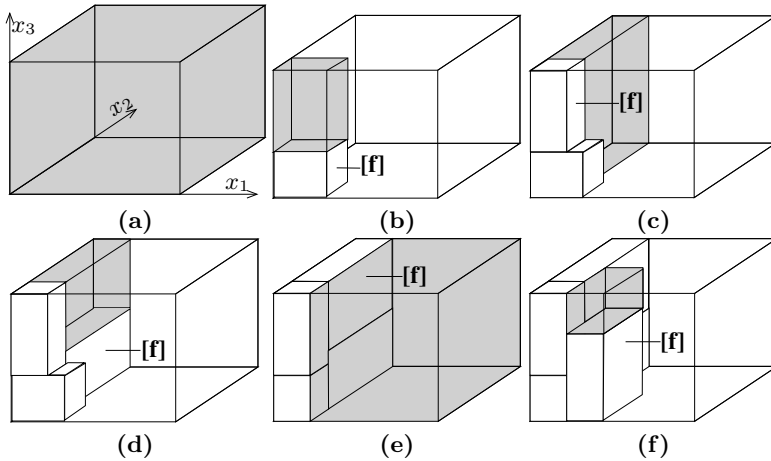


Fig. 2. Evolution of the working area. The forbidden boxes are in white and the working area in transparent (light gray). (a) At the beginning, $[\mathbf{X}] \leftarrow [\mathbf{x}]$. (b-1) $\bar{X}_1 \leftarrow \bar{f}_1$, $\bar{X}_2 \leftarrow \bar{f}_2$, $\bar{X}_3 \leftarrow \bar{f}_3$. (b-2) Since x_3 is the last dimension: $[X]_3 \leftarrow [\bar{X}_3, \bar{x}_3]$. (c-1) $\bar{X}_1 \leftarrow \bar{f}_1$ (\bar{X}_2 and \bar{X}_3 are not impacted: $\bar{f}_2 = \bar{X}_2$ and $\bar{f}_3 = \bar{X}_3$). (c-2) x_3 is saturated so $[X]_2 \leftarrow [\bar{X}_2, \bar{x}_2]$ and $[X]_3 \leftarrow [x]_3$. (d-1) only X_3 is impacted: $\bar{X}_3 \leftarrow \bar{f}_3$. (d-2) $[X]_3 \leftarrow [\bar{X}_3, \bar{x}_3]$. (e-1) Nothing is impacted. (e-2) Since x_2 is saturated, $[X]_1 \leftarrow [\bar{X}_1, \bar{x}_1]$, $[X]_2 \leftarrow [x]_2$ and $[X]_3 \leftarrow [x]_3$. (f-1) $\bar{X}_1 \leftarrow \bar{f}_1$, $\bar{X}_2 \leftarrow \bar{f}_2$, $\bar{X}_3 \leftarrow \bar{f}_3$. (f-2) $[X]_3 \leftarrow [\bar{X}_3, \bar{x}_3]$.

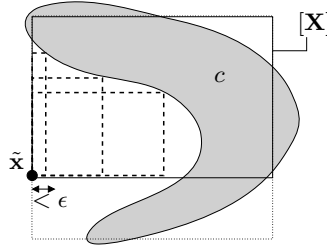


Fig. 3. Non-unicity and quality of inflation. Three different forbidden boxes (inside the same area $[X]$ and w.r.t. the same constraint c) are represented with dashed contour. They are all optimal, i.e., they cannot be inflated more in any direction. The projection of a forbidden box can be wide on a dimension and very narrow on another one.

dimensional face $[x]_{j+1} \times \dots \times [x]_n$ is covered by the projection of forbidden boxes. It is then proven that $[X]_1 \times \dots \times [X]_j$ is inconsistent. Geometrically, the following box:

$$\left([X]_1 \times \dots \times [X]_j \right) \times \left([x]_{j+1} \times \dots \times [x]_n \right) \tag{1}$$

forms a “bar”, that generalizes the “strip” in 2D above. When $j = 1$, the interval $[X]_1$ can be definitely pruned. The working area is initialized to the whole box $[x]$ and maintained in two steps each time a new box is calculated. First step: for all j , \overline{X}_j is updated in order to be the minimum of \overline{f}_j among the forbidden boxes $[f]$ obtained since the last saturation of x_{j+1} . At this point, inconsistency of (II) holds. Second step: if x_{j+1} is saturated, the area is extended to all the points necessary to saturate x_j : $[X]_j \leftarrow [\overline{X}_j, \overline{x}_j]$ and $[X]_i \leftarrow [x]_i$ for all $i > j$.

Hence, boxes are calculated in order to saturate x_n, x_{n-1}, \dots down to x_1 , each time from bottom to up. These choices are clearly arbitrary (except for x_1 that must be the last saturated variable if one is to prune this variable) and we could proceed in other ways. This means that tuples have to be ordered. The natural order we chose is the lexicographic one.

2.3 Inflaters

Let us now focus on the way forbidden boxes are built. For reasons that will be soon apparent, we consider that forbidden boxes are the result of a new kind of operator, called *inflater*. The basic idea behind an *inflater* is to take a single forbidden point $\tilde{\mathbf{x}}$ and to build a forbidden (and as large as possible) box around $\tilde{\mathbf{x}}$ inside a working area. However, it is often possible to inflate in different (and incomparable) ways as Figure 3 suggests.

Boxes that are not well-balanced slow down the sweeping process. To help the intuition, imagine that a degenerated interval (reduced to a single point) appears on a dimension. The working area is then flattened on this dimension until the subsequent dimensions are all saturated. All this work is done for no gain at all.

With a very basic geometric argument, we can say that the most extended a box is in a given direction, the less it is in another. As a consequence, taking into account the working area at the source of the inflation is important. This explains why, in the next definition, an inflater takes an initial forbidden point $\tilde{\mathbf{x}}$ and a working area $[\mathbf{X}]$ (a bad alternative would be to calculate a forbidden box unboundedly and intersect the result with the working area).

Definition 2 (Inflater). *Let \mathcal{C} be a constraint over a set of n variables, i.e., a subset of \mathbb{R}^n . An inflater I of \mathcal{C} is an operator from $\mathbb{R}^n \times \mathbb{IR}^n \rightarrow \mathbb{IR}^n$ such that:*

$$\forall [\mathbf{X}] \in \mathbb{R}^n, \forall \tilde{\mathbf{x}} \in [\mathbf{X}], \quad \begin{cases} (i) & I(\tilde{\mathbf{x}}, [\mathbf{X}]) \subseteq [\mathbf{X}] \\ (ii) & I(\tilde{\mathbf{x}}, [\mathbf{X}]) \neq \emptyset \implies \tilde{\mathbf{x}} \in I(\tilde{\mathbf{x}}, [\mathbf{X}]) \\ (iii) & I(\tilde{\mathbf{x}}, [\mathbf{X}]) \cap \mathcal{C} = \emptyset \end{cases}$$

(i) means that the inflation is bounded by the working area. (ii) means that the anchor $\tilde{\mathbf{x}}$ belongs to the inflated box. (iii) means that the inflated box is forbidden.

Note that the inflater can return an empty set. This property ensures that the definition is consistent with the case where $\tilde{\mathbf{x}}$ is feasible w.r.t. a constraint.

2.4 Algorithm

We can give now our lexicographic sweeping algorithm for pruning the lower bound of a variable. This corresponds to a revision of the algorithm *pruneMin* given in [5] page 9. The algorithm for pruning the upper bound (*pruneMax*) is easy to obtain from the same algorithm by swapping bounds and inequality signs conveniently, it will be omitted here.

The procedure *try_inflate* tries to build a forbidden box containing $\tilde{\mathbf{x}}$ and inside $[\mathbf{X}]$, by considering each constraint of \mathcal{C} in turn. The purpose of Section 3 is precisely to show how to inflate w.r.t. a given constraint. If the best forbidden box found $[\mathbf{f}]$ is significantly large, the procedure sets $[\mathbf{X}]$ to $[\mathbf{f}]$ and returns **true** (and **false** otherwise).

The changes we brought to the original discrete version are:

1. A working area is maintained instead of a *sweep point* and a *jump vector*, in the terminology of the papers cited above.
2. This working area is transmitted to the inflater for the calculation of more well-balanced forbidden boxes (as justified in §2.3).
3. Since we are in a continuous setting, the lower left corner of $[\mathbf{X}]$ is not the lowest feasible vector but the greatest unfeasible one reached by the inflater.
4. Forbidden boxes whose size is below a certain floor are discarded.

2.5 Tradeoff between Sweeping and Propagation

In presence of n variables, sweeping means dealing with a worst-case number of forbidden boxes that grows exponentially with n . Therefore, a tradeoff between sweeping and propagation is usually made. The number k of variables involved in

Algorithm 1. *pruneMin*(\mathcal{C} , $[\mathbf{x}]$, d)**Input:** a set of constraints \mathcal{C} , a k -dimensional box $[\mathbf{x}]$ **Output:** a subbox of $[\mathbf{x}]$ with the lower bound of the d^{th} coordinate pruned
w.r.t. \mathcal{C}

```

1  $b \leftarrow \mathbf{true}$ ; // true while a solution may exist in  $[\mathbf{x}]$ 
2  $success \leftarrow \mathbf{false}$ ; // true if filtering occurs
3  $[\mathbf{X}] \leftarrow [\mathbf{x}]$ ; // init the working area
4 while  $b$  and try_inflate( $\mathcal{C}$ ,  $\underline{\mathbf{X}}$ ,  $[\mathbf{X}]$ ) do // see comment below
5    $b \leftarrow \mathbf{false}$ 
6   for  $j = k - 1$  downto 0 do // for each saturated dimension
7      $j' \leftarrow (j + d) \bmod k$ 
8     if  $\overline{X}_{j'} < \overline{x}_{j'}$  then //  $(j' + 1)$  is the last saturated dimension
9       if  $j = 0$  then  $success \leftarrow \mathbf{true}$ ; // "global" filtering has occurred
10       $[X]_{j'} \leftarrow [\overline{X}_{j'}, \overline{x}_{j'}]$ ; // start a new "strip"
11       $b \leftarrow \mathbf{true}$ ; // something is left inside  $[\mathbf{x}]$ 
12      break; // exit the "for" loop ( $x_{j'}$  is not saturated)
13     $[X]_{j'} \leftarrow [x]_{j'}$ ; // reset to the whole domain (saturated dimension)
14 if  $b$  then
15    $[\mathbf{x}'] \leftarrow [\mathbf{x}]$ ; // load the initial box
16    $[\mathbf{x}']_d \leftarrow [\underline{X}_d, \overline{x}_d]$ ; // apply the global filtering
17 else
18    $[\mathbf{x}'] \leftarrow \emptyset$ ; // raise infeasibility
19 return  $[\mathbf{x}']$ 

```

a sweeping loop is typically limited to 2 or 3. The pruning obtained by sweeping is then propagated to the other groups of variables, in a classical way.

Of course, there must be some correlation between variables of a same group. Otherwise, sweeping has less effect: if we consider the extreme case of two independent variables, sweeping with these two variables amounts to filter with constraints separately, i.e., propagation.

We must therefore choose groups of k variables that are "correlated". But finding such mappings in a general setting is a problem on its own (see, e.g., [3]). This is the main reason why sweeping is well adapted to geometrical problems: we have in these problems a direct correlation between the variables that correspond to the coordinates of the same object. In other words, the dimensions of the sweeping loop match the geometrical dimensions.

3 A Generic Inflator for Arithmetical Constraints

Our inflator manipulates constraints in symbolic form. Inflation is made by an induction over the syntactical tree of the constraint expression. Symbolic processing is well known through modern computer algebra systems (like Maple or Mathematica). But it has also led to significant results in constraint programming, for building propagators and calculating enclosures of derivatives, see [7].

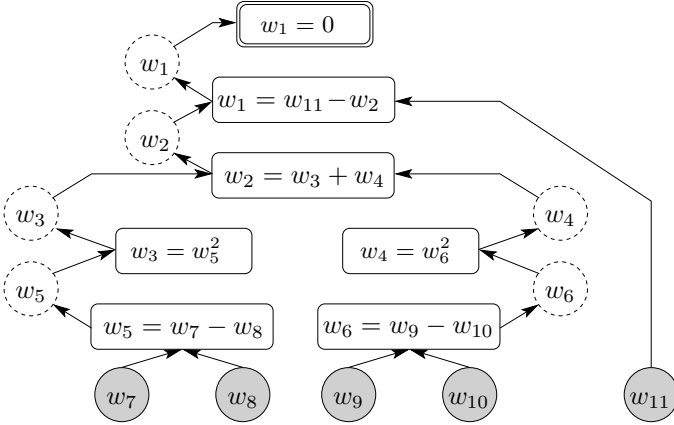


Fig. 4. Decomposition of the constraint (c) into elementary constraints. Variables w_7, \dots, w_{11} correspond to the real variables x_1, \dots, x_5 . The variables w_1, \dots, w_6 , are dummy variables, created on-the-fly upon activation of the constraint.

Our algorithm needs to symbolically inverse elementary functions, as done by HC4Revise [7] for filtering a constraint. However, the inverted functions are evaluated with a new interval arithmetic (cf. Algorithms [2] and [3]).

The syntactical tree of a mathematical expression is a composition of operators $\circ \in \{+, -, \times, /\}$ and elementary functions $\phi \in \{\text{sqr}, \text{sqrt}, \text{cos}, \dots\}$. A constraint can therefore be decomposed into a tree-structured network of elementary constraints $z = x \circ y$ and $y = \phi(x)$. For instance, consider the following distance constraint of arity 5:

$$(c) \quad x_5 = (x_1 - x_2)^2 + (x_3 - x_4)^2.$$

The network of elementary constraints equivalent to c is given in Figure [4].

Let us call $dec(c)$ the CSP resulting from the decomposition of c , and k the number of dummy variables. Let us denote by w_1, \dots, w_{k+n} the variables of $dec(c)$ sorted increasingly with respect to the partial order induced by the tree, i.e., w_1 is the variable of the root constraint and $i < j$ if w_j is a variable that appears below w_i in the same branch. Note that Figure [4] respects this convention. Let us also call $output(j)$ the index i of w such that w_i is the output variable of the elementary constraint involving w_j as input (e.g., $output(9) = 6$, $output(6) = 4$ and $output(4) = 2$). Finally, we will consider the functions $f_1(x), \dots, f_k(x)$ such that $w_i = f_i(x)$ is the constraint corresponding to the subtree below w_i . E.g., $f_5(x) = x_1 - x_2$, $f_4(x) = (x_3 - x_4)^2$.

The basic idea is to calculate a $(k + n)$ -dimensional box

$$(-\infty, +\infty)^k \times [w_{k+1}] \times \dots \times [w_{k+n}]$$

that is forbidden w.r.t. $dec(c)$. The forbidden box $[x]$ is then simply obtained by projection on the n last dimensions:

$$[\mathbf{x}] \leftarrow [w_{k+1}] \times \cdots [w_{k+n}].$$

Indeed, if some $x \in [\mathbf{x}]$ would be a solution w.r.t. c , there would be an instantiation of the w_i 's with $w_{k+1} = x_1, \dots, w_{k+n} = x_n$ that would satisfy $dec(c)$. Such instantiation would necessarily belong to $[\mathbf{f}]$, giving a contraction.

Our algorithm is split into two phases. The *forward* phase propagates $\tilde{\mathbf{x}}$ (the initial forbidden point) and $[\mathbf{X}]$, that is, calculates a point \tilde{w}_i and an interval $[W_i]$ for each variable of $dec(c)$. The *backward* phase yields the desired forbidden box.

3.1 Forward Phase

In this phase, a vector $\tilde{\mathbf{w}}$ is calculated from $\tilde{\mathbf{x}}$. Similarly, an area $[\mathbf{W}]$ is calculated from $[\mathbf{X}]$. The vector $\tilde{\mathbf{w}}$ (resp., the box $[\mathbf{W}]$) are built by instantiating leaves to $\tilde{\mathbf{x}}$ (resp. fixing their domains to $[\mathbf{X}]$) and propagating with the elementary constraints up to the root. The root constraint $w_1 = 0$ is ignored in this phase.

Base Case. For all $i > k$, set $\tilde{w}_i \leftarrow \tilde{x}_{i-k}$ and $[W_i] \leftarrow [X_{i-k}]$.

Induction. If $i < k$, w_i is the output of an elementary constraint. If the constraint is $w_i = \phi(w_j)$, we set:

$$\tilde{w}_i \leftarrow \phi(\tilde{w}_j).$$

We also set $[W_i]$ to the image of $[W_j]$ by ϕ using standard interval arithmetic:

$$[W_i] \leftarrow \phi([W_j]).$$

If the constraint is $w_i = w_{j_1} \circ w_{j_2}$, we set in a similarly way:

$$\begin{aligned} (i) \quad & \tilde{w}_i \leftarrow \tilde{w}_{j_1} \circ \tilde{w}_{j_2}, \\ (ii) \quad & [W_i] \leftarrow [W_{j_1}] \circ [W_{j_2}], \end{aligned} \tag{2}$$

At the end of the forward phase, $\tilde{\mathbf{w}}$ and $[\mathbf{W}]$ have the following properties:

$$\begin{aligned} (i) \quad & \tilde{\mathbf{w}} \text{ is unfeasible w.r.t. } dec(c), \\ (ii) \quad & \forall i, 1 \leq i \leq k+n, \quad \tilde{w}_i \in [W_i], \\ (iii) \quad & \forall i < k, f_i([\mathbf{X}]) \subseteq [W_i]. \end{aligned} \tag{3}$$

(i) comes from the fact that $\tilde{\mathbf{x}}$ is unfeasible w.r.t. c , by using the same argument by contradiction as above. (ii) and (iii) are obtained by a straightforward induction (remember that $\tilde{\mathbf{x}} \in [\mathbf{X}]$).

If it turns that $0 \notin [W_1]$, then, by (3iii), the whole box $[\mathbf{X}]$ is unfeasible. In this case, the backward phase is skipped and the area $[\mathbf{X}]$ can be directly returned.

We need now the concept of *inner inflaters* for the backward phase. We introduce it first.

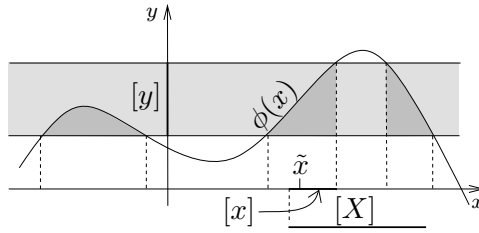


Fig. 5. Inner inflation for a function ϕ . The initial forbidden point is \tilde{x} and the working area $[X]$. The result of the inflater is the interval $[x]$.

3.2 Inner Inflaters

Consider first an elementary function ϕ . Intuitively, assume we have an interval $[y]$ that is “forbidden”, that is, values in $[y]$ are not allowed in some “context” (all this will be restated on a more rigorous footing in §3.3). The purpose of an inner inflater is to calculate an interval $[x]$ (as large as possible) such that $\phi([x]) \subseteq [y]$. Then, $[x]$ is “forbidden” since the image of all x in $[x]$ is a forbidden point of $[y]$.

However, there are often different possible intervals $[x]$, as illustrated in Figure 5. For instance, if $y = x^2$ and $[y] = [4, 9]$, then $[x]$ can be set to $[-3, -2]$ or $[2, 3]$. We will require the interval to contain a specific point \tilde{x} (a precondition being $\phi(\tilde{x}) \in [y]$). With this additional parameter, we can select a (maximal) interval containing \tilde{x} and whose image is included in $[y]$. We will also require the result to be limited by an “area” $[X]$ (again, a precondition being $\tilde{x} \in [X]$).

The exact same idea carries over binary operators. All is summarized in the following definition:

Definition 3 (Inner Inflater). *Let ϕ be an elementary function. An inner inflater of ϕ is an operator $[\phi]^\diamond$ that takes as input a point \tilde{x} and two intervals $[X]$ and $[y]$ such that $\tilde{x} \in [X]$ and $\phi(\tilde{x}) \in [y]$. Then, the result*

$$[x] := [\phi]^\diamond(\tilde{x}, [X], [y]) \text{ satisfies } \begin{cases} [x] \subseteq [X] \\ x \in [x] \\ \phi([x]) \subseteq [y] \end{cases}$$

Similarly, if \circ is an operator, an inner inflater of \circ is an operator $[\circ]^\diamond$ that takes as input a couple (\tilde{x}, \tilde{y}) and three intervals $[X], [Y]$ and $[z]$ such that $(\tilde{x}, \tilde{y}) \in [X] \times [Y]$ and $\tilde{x} \circ \tilde{y} \in [z]$. Then the result

$$[x] \times [y] := [\circ]^\diamond(\tilde{x}, \tilde{y}, [X], [Y], [z]) \text{ satisfies } \begin{cases} [x] \times [y] \subseteq [X] \times [Y] \\ (\tilde{x}, \tilde{y}) \in [x] \times [y] \\ [x] \circ [y] \subseteq [z] \end{cases}$$

Examples of algorithms for inner inflaters will be given in §3.5.

3.3 Backward Phase

The backward phase inflates $\tilde{\mathbf{w}}$ to a forbidden box $[\mathbf{f}]$ of the following form:

$$[\mathbf{f}] := (-\infty, +\infty)^k \times [w_{k+1}] \times \cdots \times [w_{k+n}]. \quad (4)$$

that also satisfies

$$(\tilde{w}_{k+1}, \dots, \tilde{w}_{k+n}) \in [w_{k+1}] \times \cdots \times [w_{k+n}] \subseteq [W_{k+1}] \times \cdots [W_{k+n}]. \quad (5)$$

To prove the correctness, we shall consider boxes $[\mathbf{f}^{(1)}], \dots, [\mathbf{f}^{(k+1)}]$ (of course, these boxes are not actually *built*). The invariant satisfied by the i^{th} box is:

$$\left\{ \begin{array}{ll} [\mathbf{f}^{(i)}] \text{ is forbidden} & (i) \\ j < i \implies [f^{(i)}]_j = (-\infty, +\infty) & (ii) \\ (j > i \wedge \text{output}(j) \geq i) \implies [f^{(i)}]_j = (-\infty, +\infty) & (iii) \\ (j \geq i \wedge \text{output}(j) < i) \implies [f^{(i)}]_j \subseteq [W]_j & (iv) \\ \forall j, \tilde{w}_j \in [f^{(i)}]_j & (v) \end{array} \right. \quad (6)$$

We proceed this time from the root down to the leaves of the tree.

Base Case. The base case is the root constraint $w_1 = 0$, $w_1 \leq 0$ or $w_1 \geq 0$. Consider $w_1 = 0$ (the other cases are derived straightforwardly). By (3ii) $\tilde{w}_1 \in [W_1]$. We have to build $[w_1] \subseteq [W_1]$ such that $\tilde{w}_1 \in [w_1]$ and $[w_1] \cap \{0\} = \emptyset$:

If $\tilde{w}_1 > 0$, we just set $[w_1]$ to $[W_1] \cap (0, +\infty)$. In practice, the open bound 0 can be replaced by any floating point number greater than 0 and smaller than \tilde{w}_1 . Similarly, if $\tilde{w}_1 < 0$, $[w_1]$ is set to $[W_1] \cap (-\infty, 0)$.

At this point, the box $[\mathbf{f}^{(1)}] := [w_1] \times (-\infty, +\infty)^{k+n-1}$ satisfies (6), as one can check directly.

Now, the intuition behind the recursion is to move the ‘‘bubble’’ (the interval $[w_1]$) downto the leaves by swapping successively the bubble with infinite intervals.

Induction. Assume now by induction that for $i > 0$, a box $[\mathbf{f}^{(i)}]$ satisfies (6).

If $i = k + 1$, we are done (w_i and the subsequent variables are the leaves) and the desired box $[\mathbf{f}]$ is simply $[\mathbf{f}^{(k+1)}]$. The conditions (4) and (5) required for $[\mathbf{f}]$ are fulfilled via (6ii), (6iv) and (6v).

If $i \leq k$, w_i is the output variable of an elementary constraint. To deal with the most general situation, we shall consider that the constraint is $w_i = w_{j_1} \circ w_{j_2}$ (and necessarily, $j_1 > i$, $j_2 > i$). The case of a binary constraint $w_i = \phi(w_{j_1})$ is obtained easily by canceling terms with j_2 in what follows.

Since $(\tilde{w}_{j_1}, \tilde{w}_{j_2}) \in [W_{j_1}] \times [W_{j_2}]$ by (3ii) and since $\tilde{w}_{j_1} \circ \tilde{w}_{j_2} = \tilde{w}_i$ by (2i) with $\tilde{w}_i \in [w_i]$ by (6v), we can apply the inner inflater of the constraint as follows:

$$[w_{j_1}] \times [w_{j_2}] \leftarrow [\circ]^\diamond(\tilde{w}_{j_1}, \tilde{w}_{j_2}, [W_{j_1}], [W_{j_2}], [w_i]).$$

Assume now that all the other components w_k ($k \notin \{i, j_1, j_2\}$) are instantiated to any values inside their respective domains $[f^{(i)}]_k$. By (6i) and (6iii) we have:

$$\forall w_i \in [w_i], \forall w_{j_1} \in (-\infty, +\infty), \forall w_{j_2} \in (-\infty, +\infty) \quad w \text{ is unfeasible.}$$

In particular,

$$\forall w_i \in [w_i], \forall w_{j_1} \in [w_{j_1}], \forall w_{j_2} \in [w_{j_2}] \quad w \text{ is unfeasible.}$$

Now, for all $(w_{j_1}, w_{j_2}) \in [w_{j_1}] \times [w_{j_2}]$, either $w_i \in [w_i]$ and w is unfeasible by virtue of the previous relation, either $w_i \notin [w_i]$ and $w_i \neq w_{j_1} \circ w_{j_2}$ (by virtue of the inner inflation) which also means that w is unfeasible. Finally:

$$\forall w_i \in (-\infty, +\infty), \forall w_{j_1} \in [w_{j_1}], \forall w_{j_2} \in [w_{j_2}] \quad w \text{ is unfeasible.} \quad (7)$$

We have “shifted the bubble” from i to j_1 and j_2 . Build $[f^{(i+1)}]$ as follows:

$$\forall k \notin \{i, j\}, [f^{(i+1)}]_k \leftarrow [f^{(i)}]_k, \quad [f^{(i+1)}]_i \leftarrow (-\infty, +\infty), \\ [f^{(i+1)}]_{j_1} \leftarrow [w_{j_1}], \quad [f^{(i+1)}]_{j_2} \leftarrow [w_{j_2}].$$

Since the choice for the other components was arbitrary in (7), we have that $[f^{(i+1)}]$ is forbidden. The other properties of (6) simply follow from the way $[f^{(i+1)}]$ was built. \square

The complexity of both phases is linear in the size of the tree (i.e., the length of the constraint expression) as long as inner inflaters take constant time. Furthermore, if inner inflaters are optimal, it can be easily proven by induction that the forbidden box is optimal in the n directions represented by the leaves of the tree. This means that the inflation w.r.t. the original constraint c is optimal if variables have no multiple occurrences. Otherwise, the best we can do is to take the union of the $[w_i]$ ’s corresponding to the same variable.

Requiring inner inflaters to be optimal and in constant time is easily fulfilled, as it will be shown below.

3.4 Remark on the Area of Dummy Variables

One may wonder why we need to restrict inflation of the w_i ’s with $i \leq k$ to an area $[W_i]$ since these variables disappear at the end of the process. Notice first that extending w_i out of the bounds of $[W_i]$ is useless because of (3.iii). Such values for w_i only have support on x outside of $[X]$. Now, by limiting in this way the inflation for a dummy variable, we can improve the inflation for another one (this will be illustrated in Figure 6) resulting, in fine, to a wider inflated box.

3.5 Algorithms for Inner Inflaters

In this section, we give two examples of inner inflaters: one for the square function and one for the addition. In order to give turn-key algorithms, we shall consider roundoff issues. All the operations are now potentially inaccurate and can be rounded upward or downward. The two variants are marked by the symbols \triangle and ∇ respectively. For instance, $x \triangleleft y$ is the addition rounded upward; $\nabla \sqrt{y}$ is the square root of y rounded downward.

Algorithm 2. $[sqr]^\diamond([\tilde{x}], [X], [y])$

Input: three intervals $[\tilde{x}]$, $[y]$ and $[X]$ with $[\tilde{x}] \subseteq [X]$ and $[\tilde{x}]^2 \subseteq [y]$
Output: a maximal interval $[x]$ such that $[\tilde{x}] \subseteq [x] \subseteq [X]$ and $[x]^2 \subseteq [y]$

```

1  $u \leftarrow \max\{\sqrt{\underline{y}}, 0\}$ ;
2 if  $\underline{y} > 0$  then
3    $l \leftarrow \lceil \sqrt{\underline{y}} \rceil$ ;
4   if  $l < u$  then
5     if  $\tilde{x} > 0$  then return  $([X] \cap [l, u]) \cup [\tilde{x}]$ ;
6     else return  $([X] \cap [-u, -l]) \cup [\tilde{x}]$ ;
7   else return  $[\tilde{x}]$ ;
8 else return  $([X] \cap [-u, u]) \cup [\tilde{x}]$ ;
```

In the forward phase above, the vector $\tilde{\mathbf{w}}$ was calculated from $\tilde{\mathbf{x}}$ by applying elementary functions and operators recursively. Because of rounding errors, all these operations are inaccurate and must be replaced by interval counterparts. Therefore, all the \tilde{w}_i 's are now intervals $[\tilde{w}_i]$. And, for the leaves, we set $[\tilde{w}_{k+i}]$ to the degenerated interval $[\tilde{x}_i, \tilde{x}_i]$. Note that rounding may cause $[\tilde{w}_1]$ to contain 0 although the initial box $[\tilde{\mathbf{x}}]$ is forbidden w.r.t. c . In this case, the backward phase can also be skipped since no inflation at all can be expected.

Square root

line 1. u represents the maximal upper bound of $[x]$. To guarantee $[x]^2 \subseteq [y]$, we must have $\bar{x}^2 \leq \bar{y}$, i.e., $\bar{x} \leq \sqrt{\bar{y}}$. Therefore, $\sqrt{\bar{y}}$ has to be rounded downward. But, at least in theory, this may lead to a negative number, whence the max.

line 2. If $\underline{y} > 0$, there are two disjoint intervals whose image is $[y]$ (e.g., if $[y] = [4, 9]$, $[-3, -2]^2 = [2, 3]^2 = [y]$). We must identify the one that contains \tilde{x} .

line 3. l represents the lowest positive bound for $[x]$ which is $\sqrt{\underline{y}}$ rounded upward.

lines 4 & 7. Because of rounding, l is not necessarily smaller than u . If $l > u$, inflation has failed and the best we can do is to return $[\tilde{x}]$.

lines 5 & 6. If $\tilde{x} > 0$, the nonnegative interval is selected and intersected with the area $[X]$. Still because of rounding, $[\tilde{x}]$ may not be included in the resulting interval. It has to be merged. The case $\tilde{x} < 0$ is symmetric.

line 8. If $l = 0$, the largest interval whose image is included in $[y]$ is $[-u, u]$. The latter must be intersected with the area $[X]$ and merged with $[\tilde{x}]$ as before.

Addition

The variables initialized in Lines 1-3 are all represented in Figure 6. Rounding is made in order to guarantee $x_1 + y_1 \geq \underline{z}$ and $x_2 + y_2 \leq \bar{z}$. All the other lines work symmetrically. They compute two vectors: l and u , the lower and upper corners of $[x] \times [y]$. The case depicted for u in Figure 6 correspond to Line 7. The upper bound for x is $\bar{Y} - \bar{z}$, except if this value exceeds \bar{X} (whence the min).

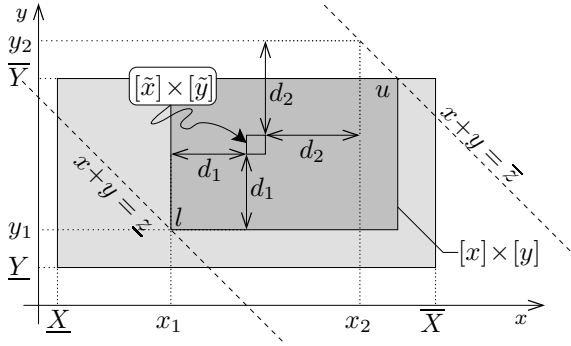


Fig. 6. Inner inflation for addition. Isolines $x+y = \underline{z}$ and $x+y = \bar{z}$ are drawn with dashes. All couple (x, y) between by these two lines satisfy $x+y \in [z]$. Therefore, the box $[\tilde{x}] \times [\tilde{y}]$ must be inflated inside this area. The half distances d_1 and d_2 gives a perfectly balanced box (a square). However, the surrounding $[X] \times [Y]$ may limit inflation in one direction, giving more freedom in the other one. This appears here with \bar{Y} being less than y_2 . The resulting box $[x] \times [y]$ is therefore a rectangle elongated on the x axis.

Algorithm 3. $[+]^\diamond(\tilde{x}, \tilde{y}, [X], [Y], [z])$

Input: 5 intervals $[\tilde{x}], [\tilde{y}], [X], [Y], [z]$ such that $[\tilde{x}] \times [\tilde{y}] \subseteq [X] \times [Y] \wedge [\tilde{x}] + [\tilde{y}] \subseteq [z]$

Output: a box $[x] \times [y]$ such that $[\tilde{x}] \times [\tilde{y}] \subseteq [x] \times [y] \subseteq [X] \times [Y]$ and $[x] + [y] \subseteq [z]$

- 1 $d_1 \leftarrow (\underline{\tilde{x}} \nabla \underline{\tilde{y}} \nabla \underline{z}) \nabla 2;$ $d_2 \leftarrow (\bar{z} \nabla \bar{\tilde{x}} \nabla \bar{\tilde{y}}) \nabla 2;$
 - 2 $x_1 \leftarrow \underline{\tilde{x}} \triangleleft d_1;$ $x_2 \leftarrow \bar{\tilde{x}} \nabla d_2;$
 - 3 $y_1 \leftarrow \underline{\tilde{y}} \triangleleft d_1;$ $y_2 \leftarrow \bar{\tilde{y}} \nabla d_2;$
 - 4 **if** $\bar{Y} > y_2$ **then** $l \leftarrow (\max\{\underline{X}, \min\{\bar{x}, \bar{z} \triangleleft \bar{Y}\}\}, \bar{Y});$
 - 5 **else if** $\underline{X} > x_1$ **then** $l \leftarrow (\underline{X}, \max\{\bar{Y}, \min\{\bar{y}, \bar{z} \triangleleft \underline{X}\}\});$
 - 6 **else** $l \leftarrow (x_1, y_1);$
 - 7 **if** $\bar{Y} < y_2$ **then** $u \leftarrow (\min\{\bar{X}, \max\{\bar{x}, \bar{z} \nabla \bar{Y}\}\}, \bar{Y});$
 - 8 **else if** $\bar{X} < x_2$ **then** $u \leftarrow (\bar{X}, \min\{\bar{Y}, \max\{\bar{y}, \bar{z} \nabla \bar{X}\}\});$
 - 9 **else** $u \leftarrow (x_2, y_2);$
 - 10 **return** $[l_1, u_1] \times [l_2, u_2];$
-

Furthermore, downward rounding may lose the upper bound of $[\tilde{x}]$ for the same reason evoked for $[\text{sqr}]^\diamond$. This bound has to be kept anyway, whence the max.

4 Discussion

In this paper, we have restated sweeping in a continuous setting and given a generic *inflater*. This inflater builds automatically forbidden boxes w.r.t. a constraint whose expression is a composition of standard mathematical operators. This makes *geost* fully generic and now applicable for packing curved objects.

The inflater is fast and optimal if no variable is duplicated in the constraint, which includes situations of particular interest such as distance constraints.

This is the first theoretical contribution for adapting sweeping (hence `geost`) to continuous domains. Of course, benchmarking has to be made now to check that our revised global constraint `geost` still outperforms generic propagation techniques in presence of curved objects. One practically useful extension of this work would also be to adapt the proposed inflater to discrete domains.

References

1. Choco: An open source Java CP library. documentation, <http://choco.emn.fr/>
2. Agren, M., Beldiceanu, N., Carlsson, M., Sbihi, M., Tuchet, C., Zampelli, S.: 6 Ways of Integreting Symmetries Within Non-Overlapping Constraints. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 11–25. Springer, Heidelberg (2009)
3. Araya, I., Trombettoni, G., Neveu, B.: Filtering Numerical CSPs Using Well-Constrained Subsystems. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 158–172. Springer, Heidelberg (2009)
4. Beldiceanu, N., Carlsson, M.: Sweep as a Generic Pruning Technique Applied to the Non-Overlapping Rectangles Constraints. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 377–391. Springer, Heidelberg (2001)
5. Beldiceanu, N., Carlsson, M., Poder, E., Sadek, R., Truchet, C.: A Generic Geometrical Constraint Kernel in Space and Time for Handling Polymorphic k-Dimensional Objects. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 180–194. Springer, Heidelberg (2007)
6. Beldiceanu, N., Carlsson, M., Thiel, S.: Sweep Synchronisation as a Global Propagation Mechanism. *Comp. & Operations Research* 33(10), 2835–2851 (2006)
7. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.-F.: Revising Hull and Box Consistency. In: ICLP, pp. 230–244 (1999)
8. Carlsson, M., Beldiceanu, N., Martin, J.: A Geometric Constraint over k-Dimensional Objects and Shapes Subject to Business Rules. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 220–234. Springer, Heidelberg (2008)
9. Carlsson, M., Ottosson, G., Carlson, B.: An open-ended finite domain constraint solver. In: 9th International Symposium on Programming Languages: Implementations, Logics, and Programs, vol. 1292, pp. 191–206 (1997)
10. Kuchcinski, K.: Constraints-driven scheduling and resource assignment. *ACM Transactions on Design Automation of Electronic Systems* 8(3), 355–383 (2003)
11. Mackworth, A.K.: Consistency in Networks of Relations. *Artificial Intelligence* 8, 99–118 (1977)

A New Hybrid Tractable Class of Soft Constraint Problems^{*}

Martin C. Cooper¹ and Stanislav Živný²

¹ IRIT, University of Toulouse III, 31062 Toulouse, France
cooper@irit.fr

² Computing Laboratory, University of Oxford, OX1 3QD Oxford, UK
standa.zivny@comlab.ox.ac.uk

Abstract. The constraint satisfaction problem (CSP) is a central generic problem in artificial intelligence. Considerable effort has been made in identifying properties which ensure tractability in such problems. In this paper we study hybrid tractability of soft constraint problems; that is, properties which guarantee tractability of the given soft constraint problem, but properties which do not depend only on the underlying structure of the instance (such as being tree-structured) or only on the types of soft constraints in the instance (such as submodularity).

We firstly present two hybrid classes of soft constraint problems defined by forbidden subgraphs in the structure of the instance. These classes allow certain combinations of binary crisp constraints together with arbitrary unary soft constraints.

We then introduce the joint-winner property, which allows us to define a novel hybrid tractable class of soft constraint problems with soft binary and unary constraints. This class generalises the `SOFTALLDIFF` constraint with arbitrary unary soft constraints. We show that the joint-winner property is easily recognisable in polynomial time and present a polynomial-time algorithm based on maximum-flows for the class of soft constraint problems satisfying the joint-winner property. Moreover, we show that if cost functions can only take on two distinct values then this class is maximal.

1 Introduction

Background. An instance of the constraint satisfaction problem (CSP) consists of a collection of variables which must be assigned values subject to specified constraints. Each CSP instance has an underlying undirected graph, known as its *constraint network*, whose vertices are the variables of the instance, and two vertices are adjacent if corresponding variables are related by some constraint. Such a graph is also known as the *structure* of the instance.

An important line of research on the CSP is to identify all tractable cases which are recognisable in polynomial time. Most of this work has been focused

^{*} Stanislav Živný gratefully acknowledges the support of EPSRC grant EP/F01161X/1, EPSRC PhD+ Award, and Junior Research Fellowship at University College, Oxford.

on one of the two general approaches: either identifying forms of constraint which are sufficiently restrictive to ensure tractability no matter how they are combined [4,15], or else identifying structural properties of constraint networks which ensure tractability no matter what forms of constraint are imposed [13].

The first approach has led to identifying certain algebraic properties known as polymorphisms [20] which are necessary for a set of constraint types to ensure tractability. A set of constraint types with this property is called a tractable *constraint language*. The second approach has been used to characterise all tractable cases of bounded-arity CSPs (such as binary CSPs): the *only* class of structures which ensures tractability (subject to certain complexity theory assumptions) are structures of *bounded tree-width* [17].

In practice, constraint satisfaction problems usually do not possess a sufficiently restricted structure or use a sufficiently restricted constraint language to fall into any of these tractable classes. Nevertheless, they may still have properties which ensure they can be solved efficiently, but these properties concern both the structure and the form of the constraints. Such properties have sometimes been called *hybrid* reasons for tractability [12,10,9,22,11].

Since in practice many constraint satisfaction problems are over-constrained, and hence have no solution, *soft* constraint satisfaction problems have been studied [12]. In an instance of the soft CSP, every constraint is associated with a function (rather than a relation as in the CSP) which represents preferences among different partial assignments, and the goal is to find the best assignment. Several very general soft CSP frameworks have been proposed in the literature [30,2]. In this paper we focus on one of the very general frameworks, the *valued* constraint satisfaction problem (VCSP) [30].

Similarly to the CSP, an important line of research on the VCSP is to identify tractable cases which are recognisable in polynomial time. It is well known that structural reasons for tractability generalise to the VCSP [12]. In the case of language restrictions, only a few conditions are known to guarantee tractability of a given set of valued constraints [8,7].

Up until now there have been very few results on hybrid tractability for the VCSP. For instance, Kumar defines an interesting framework for hybrid tractability for the Boolean weighted CSP [22]. However, to the best of our knowledge, this framework has so far not provided any new hybrid classes. In fact, all tractable classes presented in [22] are not hybrid and are already known.

Contributions. In this paper we study hybrid tractability of the VCSP. As a first step, we start with binary VCSPs. We will demonstrate two hybrid classes defined by forbidding certain graphs as induced subgraphs in the structure of the VCSP instance.¹ However, these tractable classes are not entirely satisfactory as a first step towards a general theory of hybrid tractable classes of VCSP instances since the only soft constraints they allow are unary.

As the main contribution of the paper, we introduce the *joint-winner property* (JWP), which allows us to define a *novel* hybrid tractable class of VCSPs.

¹ More precisely, in the *micro-structure complement* of the instance.

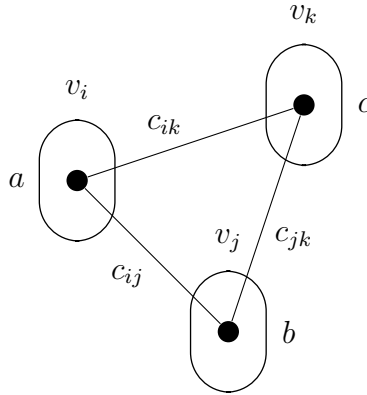


Fig. 1. The joint-winner property: $c_{ij}(a, b) \geq \min(c_{ik}(a, c), c_{jk}(b, c))$

We now describe the joint-winner property. Let v_i and v_j be two variables, and let $c_{ij}(a, b)$ denote the cost of the assignment $v_i = a$ and $v_j = b$ given by the valued constraint c_{ij} between the variables v_i and v_j . We denote by D_i the domain of variable v_i . A VCSP instance satisfies the joint-winner property if for every triple of distinct variables v_i, v_j, v_k and all domain values $a \in D_i, b \in D_j, c \in D_k$: $c_{ij}(a, b) \geq \min(c_{ik}(a, c), c_{jk}(b, c))$ (see Figure 1).

The joint-winner property is preserved by all unary constraints and hence conservative, and also easily recognisable in polynomial time. The polynomial-time algorithm for solving instances satisfying JWP is based on maximum flows.

As the next example shows, the well-known hybrid tractable class SOFT-ALLDIFF satisfies the JWP property, thus showing that JWP defines a hybrid tractable class.

Example 1. One of the most commonly used global constraints is the ALLDIFFERENT constraint [28]. Petit et al. introduced a soft version of the ALLDIFFERENT constraint, SOFTALLDIFF [27]. They proposed two types of costs to be minimised: graph- and variable-based costs. The former counts the number of equalities, whilst the latter counts the number of variables violating an ALLDIFFERENT constraint. The algorithms for filtering these constraints, introduced in the same paper, were then improved by van Hoeve et al. [33].

It is easy to check that the graph-based variant of the SOFTALLDIFF constraint satisfies the joint-winner property. In this case for every i and j , the cost function c_{ij} is defined as $c_{ij}(a, b) = 1$ if $a = b$, and $c_{ij}(a, b) = 0$ otherwise. Take any three variables v_i, v_j, v_k and $a \in D_i, b \in D_j, c \in D_k$. If $c_{ij}(a, b) = c_{jk}(b, c) = c_{ik}(a, c)$ (which means that that the domain values a, b, c are all equal or all different), then the joint-winner property is satisfied trivially. If only one of the costs is 1, then the joint-winner property is satisfied as well. Observe that due to the transitivity of equality it cannot happen that only one of the costs is 0.

In order to code the variable-based `SOFTALLDIFF` constraint as a binary VCSP \mathcal{P} , we can create n variables v'_i with domains $D_i \times \{1, 2\}$. The assignment $v'_i = (a, 1)$ means that v_i is the first variable of the original CSP to be assigned the value a , whereas $v'_i = (a, 2)$ means that v_i is assigned a but it is not the first such variable. In \mathcal{P} there is a crisp constraint which disallows $v'_i = v'_j = (a, 1)$ (for any value $a \in D_i \cap D_j$) for each pair of variables $i < j$ together with a soft unary constraint $c_i(a, k) = k - 1$ (for $k = 1, 2$) for each $i \in \{1, \dots, n\}$. Hence at most one variable can be the first to be assigned a , and each assignment of the value a to a variable (apart from the first) incurs a cost of 1. Again due to the transitivity of equality, it cannot happen that only one of the binary costs shown in the triangle of Figure 1 is zero, from which it follows immediately that the joint-winner property is satisfied in \mathcal{P} . We remark that the class defined by JWP is strictly bigger than `SOFTALLDIFF`, and hence our generic algorithm is not as efficient as tailor-made algorithms for `SOFTALLDIFF`.

When restricted to the standard CSP, the JWP property gives a set of disjoint `ALLDIFFERENT` constraints on subdomains, which is a proper (although not very surprising) generalisation of disjoint `ALLDIFFERENT` constraints.

Example 2. Suppose that n jobs must be assigned to d machines. Let $l_i(m)$ be the length of time required for machine m to complete job i . If machine m cannot perform job i , then $l_i(m) = \infty$. We use variable v_i to represent the machine to which job i is assigned. The set of jobs (which we denote by S_m) assigned to the same machine m are performed in series in increasing order of their length $l_i(m)$. The aim is to minimise T the sum, over all jobs, of their time until completion. If jobs i and j are assigned to the same machine, and $l_i(m) < l_j(m)$, then job j will have to wait during the execution of job i , contributing a time of $l_i(m)$ to the sum T . This means that

$$T = \sum_{m=1}^d \left(\sum_{i \in S_m} l_i(m) + \sum_{\substack{i, j \in S_m \\ i < j}} \min(l_i(m), l_j(m)) \right)$$

In other words, optimal assignments of jobs to machines are given by solutions to the binary VCSP with unary constraints $c_i(m) = l_i(m)$ (representing the execution times of jobs) and binary constraints

$$c_{ij}(m, m') = \begin{cases} \min(l_i(m), l_j(m)) & \text{if } m = m' \\ 0 & \text{otherwise} \end{cases}$$

(representing the waiting times of jobs).

The joint-winner property $c_{ij}(a, b) \geq \min(c_{ik}(a, c), c_{jk}(b, c))$ is trivially satisfied in this VCSP instance if a, b, c are not all equal, since in this case one of $c_{ik}(a, c), c_{jk}(b, c)$ is zero. It is also satisfied when $a = b = c = m$ since $\min(l_i(m), l_j(m)) \geq \min(\min(l_i(m), l_k(m)), \min(l_j(m), l_k(m)))$.

This problem has been shown solvable in polynomial time in [19,3].

The rest of the paper is organised as follows. In Section 2, we define binary constraint satisfaction problems (CSPs), valued constraint satisfaction problems

(VCSPs) and other necessary definitions needed throughout the paper. In Section 3, we study binary VCSPs whose only soft constraints are unary. Using a connection between these VCSPs and the maximum weighted independent set problem in certain graph classes, we identify hybrid tractable classes of VCSPs. In Section 4, we define the joint-winner property. In Section 5, we present a polynomial-time algorithm for solving binary VCSPs satisfying the joint-winner property. In Section 6, we present an important case in which this new tractable class is maximal.

2 Preliminaries

In this paper we focus on binary valued constraint satisfaction problems. We denote by \mathbb{Q}_+ the set of all non-negative rational numbers. We denote $\overline{\mathbb{Q}_+} = \mathbb{Q}_+ \cup \{\infty\}$ with the standard addition operation extended so that for all $a \in \overline{\mathbb{Q}_+}$, $a + \infty = \infty$. Members of $\overline{\mathbb{Q}_+}$ are called *costs*.

A unary cost function over domain D_i is a mapping $c_i : D_i \rightarrow \overline{\mathbb{Q}_+}$. A binary cost function over domains D_i and D_j is a mapping $c_{ij} : D_i \times D_j \rightarrow \overline{\mathbb{Q}_+}$. For notational convenience, throughout this paper we assume that there is a unique valued constraint on any given scope. In particular, $c_{ij}(a, b) = c_{ji}(b, a)$, since they are simply two different ways of representing the unique cost of simultaneously assigning $\langle a, b \rangle$ to variables $\langle i, j \rangle$. If the range of c_i (c_{ij} respectively) lies entirely within \mathbb{Q}_+ , then c_i (c_{ij} respectively) is called a *finite-valued* cost function.

If the range of c_i (c_{ij} respectively) is $\{\alpha, \infty\}$, for some $\alpha \in \mathbb{Q}_+$, then c_i (c_{ij} respectively) is called a *crisp* cost function. Note that crisp cost functions are just relations; that is, subsets of D_i (in the unary case) or $D_i \times D_j$ (in the binary case) corresponding to the set of finite-cost tuples. If c_i (c_{ij} respectively) is not a crisp cost function, it is called *soft*.

A binary VCSP instance [30] consists of a set of *variables* (denoted as v_i , where $i \in \{1, \dots, n\}$); for each variable v_i a *domain* D_i containing possible *values* for variable v_i ; and a set of *valued constraints*. Each valued constraint is either of the form $\langle v_i, c_i \rangle$, where v_i is a variable and c_i is a unary cost function (constraints of this form are called *unary* constraints), or of the form $\langle \langle v_i, v_j \rangle, c_{ij} \rangle$, where v_i and v_j are variables, the pair $\langle v_i, v_j \rangle$ is called the *scope* of the constraint, and c_{ij} is a binary cost function (constraints of this form are called *binary* constraints). A constraint is called *crisp* if its associated cost function is crisp, and similarly a constraint is called *soft* if its associated cost function is soft.

A *solution* to a VCSP instance is an assignment of values from the domains to the variables with the minimum total cost given by

$$\sum_{i=1}^n c_i(v_i) + \sum_{1 \leq i < j \leq n} c_{ij}(v_i, v_j).$$

A VCSP instance with only crisp constraints is called a CSP instance. In the CSP, the task of finding an optimal solution amounts to deciding whether there is a solution with finite cost (all constraints are satisfied).

The *microstructure* of a binary CSP instance \mathcal{P} is a graph where the set of vertices corresponds to the set of possible assignments of values to variables: a vertex $\langle v_i, a \rangle$ represents the assignment of value a to variable v_i [21]. The edges of the microstructure connect all pairs of variable-value assignments that are allowed by the constraints. (Note that if there is no explicit constraint between two variables, then it is considered to be the complete constraint.) The microstructure of a binary VCSP instance is defined similarly, but both vertices and edges of the graph are assigned costs. For CSPs, the *microstructure complement* is the complement of the microstructure: its edges represent pairs of variable-value assignments that are disallowed by the constraints. Hence for every variable v_i , the microstructure complement contains all edges of the form $\{\langle v_i, a \rangle, \langle v_i, b \rangle\}$ for $a \neq b \in D_i$ as every variable can be assigned only one value.

A *clique* in a graph is a set of vertices which are pairwise adjacent. An *independent* set in a graph is a set of vertices which are pairwise non-adjacent. It is well known that solving a CSP instance \mathcal{P} is equivalent to finding a clique of size n in the microstructure of \mathcal{P} , and to finding an independent set of size n in the microstructure complement of \mathcal{P} [21]. Therefore, tractability results on the maximum independent set problem for various classes of graphs can be easily used to obtain tractable CSP classes [10].

3 VCSPs with Crisp Binary Constraints

In this section we restrict our attention to binary VCSP instances with crisp binary constraints. There are no restrictions on unary constraints; hence both crisp and soft unary constraints are allowed. First we show how tractability results on the maximum weighted independent set in graphs can be used to obtain tractable classes of VCSPs. Next we show that the recently discovered hybrid CSP class defined by the broken-triangle property [11] is not extendible to soft unary constraints.

Let G be a graph $G = \langle V, E \rangle$ with weights $w : V \rightarrow \mathbb{Q}_+$ on the vertices of G . The weight of an independent set S in G , denoted $w(S)$, is the sum of weights of the vertices in S : $w(S) = \sum_{v \in S} w(v)$. It is easy to see that given a binary VCSP instance \mathcal{P} where only unary constraints can be soft, \mathcal{P} can be solved by finding a maximum weighted independent set in the microstructure complement of \mathcal{P} with weights given by $w(\langle v_i, a \rangle) = Mn - c_i(a)$, where M is strictly greater than the maximum finite unary cost $c_i(a)$. Indeed, independent sets of weight strictly greater than $Mn(n - 1)$ are in one-to-one correspondence with assignments to n variables in \mathcal{P} .

Two well-studied classes of graphs for which the maximum weighted independent set problem (WIS) is known to be solvable in polynomial time are perfect graphs and claw-free graphs.

A graph G is *perfect* if for every induced subgraph H of G , the chromatic number (the smallest number of colours needed to colour vertices of H such that adjacent vertices are coloured differently) of H is equal to the size of the largest clique in H . The Strong Perfect Graph Theorem states that a graph G is perfect

if, and only if, G does not contain any cycle of odd length ≥ 5 (known as a hole) nor any complement of a cycle of odd length ≥ 5 (known as an antihole) as an induced subgraph [6]. It is known that WIS in perfect graphs is solvable in polynomial time [18]. Moreover, perfect graphs can be recognised in polynomial time [5].

A graph G is *claw-free* if it does not contain a claw as an induced subgraph, where a claw is a complete bipartite graph $K_{1,3}$ with 1 vertex in one group and 3 vertices in the other group. It is obvious that claw-free graphs are recognisable in polynomial time. Extending Edmond's algorithm for maximum matchings in graphs [14], Minty designed an algorithm for the independent set problem in claw-free graphs [24]. Minty's algorithm was later corrected and extended to the maximum weighted independent set problem in claw-free graphs [25]. The combination of these results gives:

Theorem 1. *The class of VCSP instances (with crisp binary and arbitrary unary constraints) whose microstructure complement is either perfect or claw-free is tractable.*

The tractability of VCSPs with perfect microstructure (complement) and soft unary constraints was also pointed out by Takhanov [32].

Next we show that the recently discovered hybrid class of tractable CSPs defined by the broken-triangle property [11] is *not* extendible to VCSPs with soft unary constraints. A binary CSP instance \mathcal{P} satisfies the broken-triangle property with respect to the variable ordering $<$ if, and only if, for all triples of variables v_i, v_j, v_k such that $i < j < k$, if $c_{ij}(u, v) < \infty$, $c_{ik}(u, a) < \infty$ and $c_{jk}(v, b) < \infty$, then either $c_{ik}(u, b) < \infty$ or $c_{jk}(v, a) < \infty$. (In other words, every "broken" triangle $a - u - v - b$ can be closed.)

Let $\langle G, k \rangle$ be an instance of the decision version of the maximum independent set problem which consists in deciding whether there is an independent set of size at least k in graph G . This problem is known to be NP-complete [16]. We now transform this instance into a binary VCSP instance with soft unary constraints that satisfies the broken-triangle property.

Every vertex of G is represented by a Boolean variable v_i where $D_i = \{0, 1\}$. We impose the constraint $\langle \langle v_i, v_j \rangle, \{ \langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle \} \rangle$ if the vertices corresponding to v_i and v_j are adjacent in G . Now the assignments satisfying all constraints are in one-to-one correspondence with independent sets I in G , where vertex $i \in I$ if and only if $v_i = 1$. We also impose the soft unary constraints $\langle v_i, c_i \rangle$, where $c_i(x) = 1 - x$. The unary constraints ensure that the goal is to minimise the number of variables assigned value 0, which is the same as maximising the number of variables assigned value 1. Therefore, the constructed VCSP instance is equivalent to the given maximum independent set problem. It remains to show that the resulting VCSP instance satisfies the broken-triangle property with respect to some ordering. In fact, we show that it is satisfied with respect to any ordering. Take any three variables v_i, v_j, v_k . If either of the pairs of variables $\langle v_i, v_k \rangle, \langle v_j, v_k \rangle$ are not constrained, then the broken-triangle property is trivially satisfied. Assume therefore that these two constraints are present. The situation is illustrated in Figure 2. It can be checked easily that

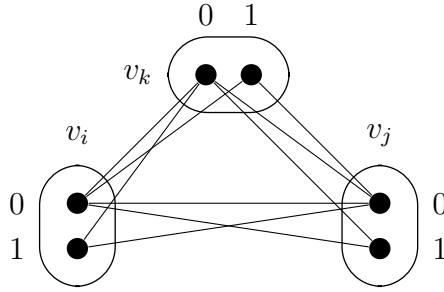


Fig. 2. VCSP encoding maximum independent set

the broken-triangle property is indeed satisfied whether the constraint on $\langle v_i, v_j \rangle$ is $\{(0, 0), \langle 0, 1 \rangle, \langle 1, 0 \rangle\}$ (as shown in Figure 2) or the complete constraint. This gives us the following result:

Theorem 2. *Assuming $P \neq NP$, the broken-triangle property cannot be extended to a tractable class including soft unary constraints.*

4 Joint-Winner Property

In this section we define the joint-winner property (see Figure 1), which is the key concept in this paper. We also study basic properties of VCSPs satisfying the joint-winner property as these will be important in designing a polynomial-time algorithm in Section 5.

Definition 1 (Joint-winner property). *A triple of variables $\langle v_i, v_j, v_k \rangle$ satisfies the joint-winner property (JWP) if $c_{ij}(a, b) \geq \min(c_{ik}(a, c), c_{jk}(b, c))$ for all domain values $a \in D_i, b \in D_j, c \in D_k$.*

A binary VCSP instance satisfies the joint-winner property if every triple of distinct variables of the instance satisfies the joint-winner property.

The joint-winner property places no restrictions on the unary soft constraints c_i . Note that the JWP on a CSP instance amounts to forbidding the multiset of binary costs $\{\alpha, \infty, \infty\}$ (for $\alpha < \infty$) in any triangle formed by three assignments to distinct variables. Since this combination can never occur on triples of variables $\langle v_i, v_j, v_k \rangle$ constrained by the three binary constraints $v_i \neq v_j \neq v_k \neq v_i$, the class of CSPs satisfying the joint-winner property generalises the ALLDIFFERENT constraint with arbitrary soft unary constraints.

The next lemma explains the reason for the name of the joint-winner property: in every triangle there is no unique smallest cost.

Lemma 1. *A VCSP instance satisfies the joint-winner property if and only if, for all triples of distinct variables $\langle v_i, v_j, v_k \rangle$ and for all $a \in D_i, b \in D_j, c \in D_k$, two of the costs $c_{ij}(a, b), c_{ik}(a, c), c_{jk}(b, c)$ are equal and less than or equal to the third.*

Proof. Assume that the joint-winner property is satisfied on the triples of variables $\langle v_i, v_j, v_k \rangle$, $\langle v_j, v_k, v_i \rangle$ and $\langle v_k, v_i, v_j \rangle$, and write $\alpha = c_{ij}(a, b)$, $\beta = c_{ik}(a, c)$ and $\gamma = c_{jk}(b, c)$. Without loss of generality, let $\alpha = \min(\alpha, \beta, \gamma)$. From $\alpha \geq \min(\beta, \gamma)$, we can deduce that $\alpha = \min(\beta, \gamma)$ and hence that two of α, β, γ are equal and less than or equal to the third.

On the other hand, if two of α, β, γ are equal and less than or equal to the third, then $\min(\beta, \gamma) = \min(\alpha, \beta, \gamma) \leq \alpha$ and the JWP is satisfied. \square

Lemma 2. *Let \mathcal{P} be a binary VCSP instance. Then, for a fixed α , the edges of the microstructure of \mathcal{P} corresponding to binary costs of at least α , together with the corresponding vertices, form non-intersecting cliques.*

Proof. For a contradiction let us assume that the edges of the microstructure of \mathcal{P} corresponding to binary costs of at least α do not form non-intersecting cliques. This means that there are three vertices $\langle v_i, a \rangle, \langle v_j, b \rangle, \langle v_k, c \rangle$ of the microstructure such that $c_{ij}(a, b) \geq \alpha$, $c_{ik}(a, c) \geq \alpha$, and $c_{jk}(b, c) < \alpha$. But this is in contradiction with Lemma 1. \square

Lemma 3. *Let \mathcal{P} be a binary VCSP instance. Let C_α be a clique in the microstructure of \mathcal{P} corresponding to binary costs of at least α , and C_β a clique in the microstructure of \mathcal{P} corresponding to binary costs of at least β . If C_α intersects C_β and $\alpha \leq \beta$, then $C_\alpha \supseteq C_\beta$.*

Proof. Suppose that C_α and C_β intersect and $\alpha \leq \beta$. If $\alpha = \beta$, the claim is satisfied trivially by Lemma 2, so we can suppose that $\alpha < \beta$. For a contradiction, assume that $C_\alpha \not\supseteq C_\beta$. By our assumptions, $\exists \langle v_i, a \rangle \in C_\alpha \cap C_\beta$ and $\exists \langle v_j, b \rangle \in C_\beta \setminus C_\alpha$. Since C_β is a clique, we must have $c_{ij}(a, b) \geq \beta > \alpha$. Thus, by Lemma 2, the edge $\{\langle v_j, b \rangle, \langle v_i, a \rangle\}$ is part of a clique C'_α of edges of cost at least α (but not C_α since $\langle v_j, b \rangle \notin C_\alpha$). But then C_α and C'_α intersect at $\langle v_i, a \rangle$ which contradicts Lemma 2. \square

5 Algorithm

In this section we present a polynomial-time algorithm for solving binary VCSPs satisfying the joint-winner property. The algorithm is an extension of a reduction to the standard max-flow/min-cut problem that has been used for flow-based soft global constraints [29,33,23].

First we review some basics on flows in graphs (see [1] for more details). Let $G = (V, A)$ be a directed graph with vertex set V and arc set A . To each arc $a \in A$ we assign a *demand/capacity* function $[d(a), c(a)]$ and a *weight* function $w(a)$. Let $s, t \in V$. A function $f : A \rightarrow \mathbb{Q}$ is called an $s - t$ *flow* (or a flow) if

- $f(a) \geq 0$ for all $a \in A$;
- for all $v \in V \setminus \{s, t\}$, $\sum_{a=(u,v) \in A} f(a) = \sum_{a=(v,u) \in A} f(a)$ (flow conservation).

We say that a flow is *feasible* if $d(a) \leq f(a) \leq c(a)$ for each $a \in A$. We define the *value* of flow f as $val(f) = \sum_{a=(s,v) \in A} f(a) - \sum_{a=(v,s) \in A} f(a)$. We define the

cost of flow f as $\sum_{a \in A} w(a)f(a)$. A *minimum-cost flow* is a feasible flow with minimum cost.

Algorithms for finding the minimum-cost flow of a given value are described in [31, Chapter 12] and [1]. Given a network G with integer demand and capacity functions, the *successive shortest path algorithm* [31], can be used to find a feasible $s - t$ flow with value α and minimum cost in time $O(\alpha \cdot SP)$, where SP is the time to compute a shortest directed path in G .

Given a VCSP \mathcal{P} satisfying the JWP, we construct a directed \mathcal{P} graph $G_{\mathcal{P}}$ whose minimum-cost integral flows of value n are in one-to-one correspondence with the solutions to \mathcal{P} . Apart from the source node s , $G_{\mathcal{P}}$ has three types of node:

1. a variable node v_i ($i = 1, \dots, n$) for each variable of \mathcal{P} ;
2. an assignment node $\langle v_i, a \rangle$ ($a \in D_i, i = 1, \dots, n$) for each possible variable-value assignment in \mathcal{P} ;
3. a clique node C_α for each clique of edges in the microstructure of \mathcal{P} corresponding to binary costs of at least α . (The subscript α is equal to the minimum cost of edges in the clique and, where necessary, we use $C_\alpha, C'_\alpha, \dots$ to denote the distinct non-intersecting cliques corresponding to the same value of α .)

In $G_{\mathcal{P}}$ there is an arc (s, v_i) for each variable v_i of \mathcal{P} . For all variables v_i and for each $a \in D_i$, there is an arc $(v_i, \langle v_i, a \rangle)$ and also an arc from $\langle v_i, a \rangle$ to the clique C_α containing $\langle v_i, a \rangle$ such that α is maximal (C_α is unique by Lemma 3).

We say that clique C_β is the *father* of clique C_α if it is the minimal clique which properly contains C_α , i.e. $C_\alpha \subset C_\beta$ (and hence $\alpha > \beta$) and $\nexists C_\gamma$ such that $C_\alpha \subset C_\gamma \subset C_\beta$ (C_β is unique by Lemma 3). In $G_{\mathcal{P}}$, for each clique C_α with father C_β , there is a bundle of arcs from C_α to C_β consisting of r arcs e_i ($i = 1, \dots, r$), where r is the number of vertices in the clique C_α . The weight of arc e_i from C_α to C_β is $w(e_i) = i(\alpha - \beta)$. (If $\alpha = \infty$ then there is a single arc of weight 0; the arcs of weight ∞ can simply be omitted.) We identify the sink node t with the clique C_0 consisting of all edges in the microstructure (since all binary costs are at least 0).

Each arc has demand 0 and capacity 1 except for arcs (s, v_i) which have both demand 1 and capacity 1 (this forces a flow of exactly 1 through each variable node v_i). Weights of all arcs are 0 except for arcs going from a clique node to its father clique node, as described above, and arcs from a variable node v_i to an assignment node $\langle v_i, a \rangle$ which have a weight of $c_i(a)$.

We show below that integral flows in the constructed network are in one-to-one correspondence with assignments in \mathcal{P} , but first we give an example.

Example 3. We show the general construction on a simple example. Let \mathcal{P} be a VCSP instance with variables v_1, v_2, v_3 and $D_1 = D_2 = \{a, b\}$, $D_3 = \{a\}$. The microstructure of \mathcal{P} is shown in Figure 3. Missing edges have cost 0. There are two cliques corresponding to cost at least 1: $C_1 = \{\langle v_1, a \rangle, \langle v_2, a \rangle, \langle v_3, a \rangle\}$ and $C'_1 = \{\langle v_1, b \rangle, \langle v_2, b \rangle\}$; and one clique corresponding to cost at least 2: $C_2 = \{\langle v_1, a \rangle, \langle v_2, a \rangle\}$ (see Figure 3). The network corresponding to instance \mathcal{P} is shown in Figure 4; demands and capacities are in square brackets, and

weights of arcs without numbers are 0. The bold edges represent flow f corresponding to the assignment $v_1 = v_2 = v_3 = a$ with total cost 4, which is the same as the cost of f .

We now prove that integral flows f in $G_{\mathcal{P}}$ are in one-to-one correspondence with assignments in the VCSP \mathcal{P} and, furthermore, that the cost of f is equal to the cost in \mathcal{P} of the corresponding assignment.

All feasible flows have value n since all n arcs (s, v_i) leaving the source have both demand and capacity equal to 1. Integral flows in $G_{\mathcal{P}}$ necessarily correspond to the assignment of a unique value a_i to each variable v_i since the flow of 1 through node v_i must traverse a node $\langle v_i, a_i \rangle$ for some unique $a_i \in D_i$. It remains to show that for every assignment $\langle a_1, \dots, a_n \rangle$ to $\langle v_1, \dots, v_n \rangle$ which is feasible (i.e. whose cost in \mathcal{P} is finite), there is a corresponding minimum-cost integral feasible flow f in $G_{\mathcal{P}}$ of cost $\sum_i c_i(a_i) + \sum_{i < j} c_{ij}(a_i, a_j)$.

For each arc a which is incoming to or outgoing from $\langle v_i, u \rangle$ in $G_{\mathcal{P}}$, let $f(a) = 1$ if $u = a_i$ and 0 otherwise. We denote the number of assignments $\langle v_i, a_i \rangle$ in clique C_α by $N(C_\alpha) = |\{\langle v_i, a_i \rangle \in C_\alpha : 1 \leq i \leq n\}|$. By construction, each clique node C_α in $G_{\mathcal{P}}$ only has outgoing arcs to its father clique. For the outgoing arc a of weight i from C_α to its father clique, let $f(a) = 1$ if $N(C_\alpha) > i$ and 0 otherwise. This simply means that the outgoing arcs (each of capacity 1) from C_α are used in increasing order of their weight, one per assignment $\langle v_i, a_i \rangle \in C_\alpha$. This is clearly a minimum-cost flow corresponding to the assignment $\langle a_1, \dots, a_n \rangle$.

Let $cf(C_\alpha)$ denote the cost β of the father clique C_β of C_α . The cost of flow f is given by

$$\begin{aligned} & \sum_{i=1}^n c_i(a_i) + \sum_{C_\alpha} \sum_{i=1}^{N(C_\alpha)-1} i(\alpha - cf(C_\alpha)) \\ &= \sum_{i=1}^n c_i(a_i) + \sum_{C_\alpha} \frac{(N(C_\alpha) - 1)N(C_\alpha)}{2}(\alpha - cf(C_\alpha)) \end{aligned}$$

This corresponds precisely to the cost of the assignment $\langle a_1, \dots, a_n \rangle$ in \mathcal{P} , since in a clique C_α with father clique C_β , each of the $(N(C_\alpha) - 1)N(C_\alpha)/2$ binary constraints contributes a cost of $\alpha - \beta$ over and above the cost of β for each of the edges in C_β .

Theorem 3. *VCSPs satisfying the joint-winner property are recognisable and solvable in polynomial time.*

Proof. From Definition [1](#), recognition can be achieved in $O(n^3 d^3)$ time, where $d = \max_{1 \leq i \leq n} |D_i|$ is the size of the largest domain.

To solve a VCSP satisfying the JWP, we create a vertex for each of the cliques corresponding to binary costs of at least α . There are at most $|D_i| \times |D_j|$ different costs in the cost function c_{ij} . Hence in total there are at most $O(n^2 d^2)$ different cliques. So our network has $O(n^2 d^2 + nd + n + 2) = O(n^2 d^2)$ vertices. The result follows from the fact that a polynomial-time algorithm exists for determining a minimum-cost maximum flow in a network. In particular, using the successive

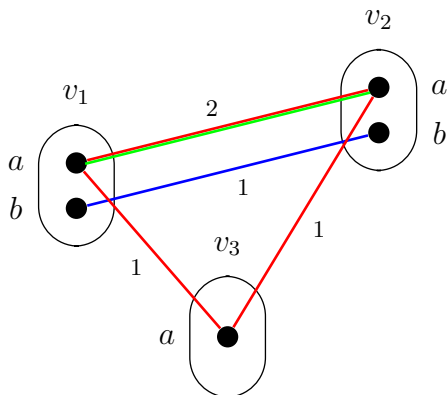


Fig. 3. Microstructure of \mathcal{P} described in Example 3

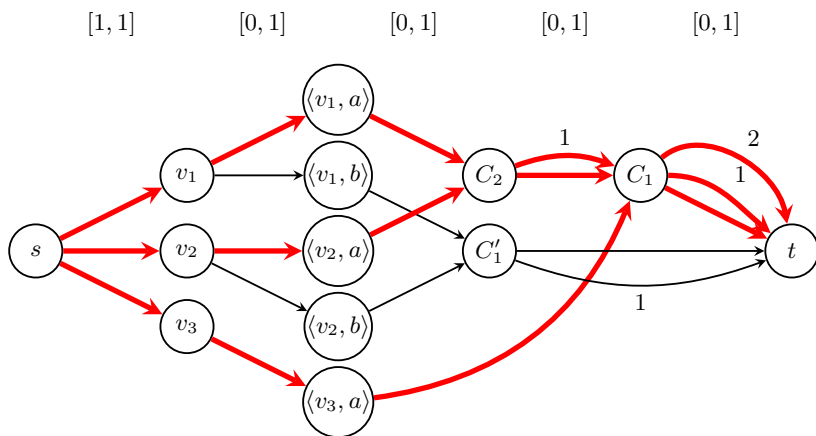


Fig. 4. Network $G_{\mathcal{P}}$ corresponding to the VCSP \mathcal{P} of Example 3

shortest path algorithm, the running time is $O(n \cdot SP)$, where SP is the time to compute a shortest directed path in the network [31][1]. Using Fibonacci heaps, this is $O(n(n^4 d^4 + n^2 d^2 \log(n^2 d^2))) = O(n^5 d^4)$. \square

6 Maximality

Tractable classes defined by structural or language restrictions are often shown to be maximal. That is, any extension of the class is NP-hard. We consider that a hybrid tractable class defined by a set of possible combinations of costs within a subproblem is maximal if extending it to include any other single combination of costs renders the problem NP-hard. In particular, since JWP is defined on 3-variable subproblems, we call an instance *3-maximal* if extending it to include any other single combination of costs on 3 variables renders the problem NP-hard. The existence of a larger tractable class subsuming JWP and defined by a rule on k -variable subproblems (for $k > 3$) is an interesting open question.

In this section we show a special case for which the joint-winner property is 3-maximal, namely when all binary cost functions take on only two possible costs $\alpha < \beta$.

Theorem 4. *If all costs belong to $\{\alpha, \beta\}$ (for some fixed distinct costs $\alpha < \beta$), then the joint-winner property defines a 3-maximal tractable class provided $d > 2$ or $(d \geq 2) \wedge (\beta < \infty)$, where d is the maximum domain size.*

Proof. To prove 3-maximality we have to show the NP-hardness of the set of instances defined by the fact that in each triangle the triple of costs either satisfies the joint-winner property or is just one other fixed combination. Since all costs belong to $\{\alpha, \beta\}$ where $\alpha < \beta$, from Definition [1] the only situation forbidden by the JWP is that there are 3 variables v_i, v_j, v_k and domain values $a \in D_i, b \in D_j, c \in D_k$ such that $c_{ij}(a, b) = \alpha$ and $c_{ik}(a, c) = c_{jk}(b, c) = \beta$. Hence extending the JWP means allowing all combinations of costs from $\{\alpha, \beta\}$ in all triangles.

If $\beta = \infty$, allowing all combinations of costs means that our instance allows all binary relations (corresponding to the set of finite-cost tuples) and hence we can encode any binary CSP. This is NP-complete if $d > 2$.

If $\beta < \infty$, allowing all combinations of costs in $\{\alpha, \beta\}$ is equivalent to the set of instances of MAX-CSP in which no two constraints can have the same scope. The NP-hardness of this latter problem for $d \geq 2$ follows from the following reduction from MAX-CSP [26]. A polynomial reduction of an instance I of MAX-CSP into an equivalent instance I' in which no two constraints have the same scope can be achieved by replacing each variable v_i in I by M variables v_i^j ($j = 1, \dots, M$) in I' constrained by a clique of equality constraints, where M is greater than the total number of constraints in I . In the optimal solution to I' , variables v_i^j ($j = 1, \dots, M$) are necessarily assigned the same value (otherwise a cost of at least M would be incurred). \square

7 Conclusions

We consider the tractable class of VCSPs defined by the joint-winner property (JWP) as a necessary first step towards a general theory of tractability of optimisation problems which will eventually cover structural, language and hybrid reasons for tractability.

The JWP is interesting in its own right since it is a proper extension to known tractable classes (such as VCSPs consisting of arbitrary unary constraints and non-intersecting `SOFTALLDIFF` constraints of arbitrary arity).

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Pearson (2005)
2. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based Constraint Satisfaction and Optimisation. *Journal of the ACM* 44(2), 201–236 (1997)
3. Bruno, J.L., Coffman, E.G., Sethi, R.: Scheduling Independent Tasks to Reduce Mean Finishing Time. *Communications of the ACM* 17(7), 382–387 (1974)
4. Bulatov, A., Krokhin, A., Jeavons, P.: Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing* 34(3), 720–742 (2005)
5. Chudnovsky, M., Cornuéjols, G., Liu, X., Seymour, P.D., Vušković, K.: Recognizing Berge graphs. *Combinatorica* 25(2), 143–186 (2005)
6. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. *Annals of Mathematics* 164(1), 51–229 (2006)
7. Cohen, D.A., Cooper, M.C., Jeavons, P.G.: Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theoretical Computer Science* 401(1-3), 36–51 (2008)
8. Cohen, D.A., Cooper, M.C., Jeavons, P.G., Krokhin, A.A.: The Complexity of Soft Constraint Satisfaction. *Artificial Intelligence* 170(11), 983–1016 (2006)
9. Cohen, D., Jeavons, P.: The complexity of constraint languages. In: Rossi, F., van Beek, P., Walsh, T. (eds.) *Handbook of Constraint Programming*. Elsevier, Amsterdam (2006)
10. Cohen, D.A.: A New Class of Binary CSPs for which Arc-Consistency Is a Decision Procedure. In: Rossi, F. (ed.) *CP 2003*. LNCS, vol. 2833, pp. 807–811. Springer, Heidelberg (2003)
11. Cooper, M.C., Jeavons, P.G., Salamon, A.Z.: Generalizing constraint satisfaction on trees: hybrid tractability and variable elimination. *Artificial Intelligence* (2010)
12. Dechter, R.: *Constraint Processing*. Morgan Kaufmann, San Francisco (2003)
13. Dechter, R., Pearl, J.: Network-based Heuristics for Constraint Satisfaction Problems. *Artificial Intelligence* 34(1), 1–38 (1988)
14. Edmonds, J.: Paths, trees, and flowers. *Canad. J. Math.* 17, 449–467 (1965)
15. Feder, T., Vardi, M.: The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing* 28(1), 57–104 (1998)
16. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York (1979)
17. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM* 54(1) (2007)

18. Grötschel, M., Lovasz, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1(2), 169–198 (1981)
19. Horn, W.A.: Minimizing Average Flow Time with Parallel Machines. *Operations Research* 21(3), 846–847 (1973)
20. Jeavons, P.: On the Algebraic Structure of Combinatorial Problems. *Theoretical Computer Science* 200(1-2), 185–204 (1998)
21. Jégou, P.: Decomposition of Domains Based on the Micro-Structure of Finite Constraint-Satisfaction Problems. In: *AAAI*, pp. 731–736 (1993)
22. Kumar, T.K.S.: A framework for hybrid tractability results in boolean weighted constraint satisfaction problems. In: Stuckey, P.J. (ed.) *CP 2008*. LNCS, vol. 5202, pp. 282–297. Springer, Heidelberg (2008)
23. Lee, J.H.M., Leung, K.L.: Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In: *IJCAI*, pp. 559–565 (2009)
24. Minty, G.J.: On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B* 28(3), 284–304 (1980)
25. Nakamura, D., Tamura, A.: A revision of Minty’s algorithm for finding a maximum weighted stable set of a claw-free graph. *J. Oper. Res. Soc.* 44(2), 194–204 (2001)
26. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley, Reading (1994)
27. Petit, T., Régim, J.C., Bessière, C.: Specific Filtering Algorithms for Over-Constrained Problems. In: Walsh, T. (ed.) *CP 2001*. LNCS, vol. 2239, pp. 451–463. Springer, Heidelberg (2001)
28. Régim, J.C.: A filtering algorithm for constraints of difference in CSPs. In: *AAAI*, pp. 362–367 (1994)
29. Régim, J.C.: Cost-based arc consistency for global cardinality constraints. *Constraints* 7(3-4), 387–405 (2002)
30. Schiex, T., Fargier, H., Verfaillie, G.: Valued Constraint Satisfaction Problems: Hard and Easy Problems. In: *IJCAI*, pp. 631–637 (1995)
31. Schrijver, A.: *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics, vol. 24. Springer, Heidelberg (2003)
32. Takhanov, R.: A Dichotomy Theorem for the General Minimum Cost Homomorphism Problem. In: *STACS*, pp. 657–668 (2010)
33. van Hoeve, W.J., Pesant, G., Rousseau, L.M.: On global warming: Flow-based soft global constraints. *J. Heuristics* 12(4-5), 347–373 (2006)

A Propagator for Maximum Weight String Alignment with Arbitrary Pairwise Dependencies

Alessandro Dal Palù¹, Mathias Möhl², and Sebastian Will^{2,3}

¹ Dipartimento di Matematica, Università degli Studi di Parma, Parma, Italy
alessandro.dalpalu@unipr.it

² Bioinformatics, Institute of Computer Science, Albert-Ludwigs-Universität,
Freiburg, Germany

{mmohl,will}@informatik.uni-freiburg.de

³ Computation and Biology Lab, CSAIL, MIT, Cambridge MA, USA
swill@csail.mit.edu

Abstract. The optimization of weighted string alignments is a well studied problem recurring in a number of application domains and can be solved efficiently. The problem becomes MAX-SNP-hard as soon as arbitrary pairwise dependencies among the alignment edges are introduced. We present a global propagator for this problem which is based on efficiently solving a relaxation of it. In the context of bioinformatics, the problem is known as alignment of arc-annotated sequences, which is e.g. used for comparing RNA molecules. For a restricted version of this alignment problem, we show that a constraint program based on our propagator is on par with state of the art methods. For the general problem with unrestricted dependencies, our tool constitutes the first available method with promising applications in this field.

The maximum weight string alignment problem for strings S_a and S_b asks for a partial matching of positions in S_a and S_b that preserves the string order and has maximum weight. This problem is efficiently solved by dynamic programming (DP) [10]. An extended variant of the problem introduces pairwise dependencies among positions in each string S_a and S_b . In this problem variant, one optimizes the sum of weights, where a weight is associated with each pair of matched positions and with each pair of matched dependencies. In general, this problem is MAX-SNP-hard [2].

In bioinformatics, the problem has been studied as alignment of arc-annotated sequences, where each arc represents a dependency. There, the problem has applications in aligning RNA or protein molecules that can be abstracted as sequences of monomers and structural dependencies among those that represent a proximity of the respective positions in the molecules structure.

Due to the hardness of the problem, restricted versions with limited dependencies, in particular nested and crossing ones, have been considered. For nested and also certain restricted crossing dependencies the alignment problem can be solved efficiently with DP algorithms [6,9]. Heuristic approaches based on Integer

Linear Programming (ILP) are available for crossing dependencies specialized to RNA [1] and for unlimited dependencies specialized to proteins [3]. By using DP-based propagation, our approach is similar to Trick [11]. Furthermore, there is prior work applying this idea in an optimization setting [5].

Contribution. In this work, we consider a constraint programming approach to the arc-annotated sequence alignment problem with unlimited dependencies. Our main contribution is a general propagator for the maximum weight string alignment with arbitrary pairwise dependencies. The propagator is based on a relaxation that resolves the dependencies by bounding their weight contribution. It propagates on the upper bound of the total weight and prunes the valid alignments accordingly. Furthermore, we discuss decomposition into independent subproblems for optimization approaches using our propagator.

We apply the technique to RNA sequence-structure alignment and show that results are comparable to the state-of-the-art ILP approach Lara [1]. Finally, we discuss a new method to compare riboswitches, which was not applicable before, because prior RNA alignment approaches score at most crossing dependencies.

1 Preliminaries

An *arc-annotated sequence* is a pair (S, P) , where the *sequence* S is a string over some alphabet Σ and P is a set of *arcs* (i, j) with $1 \leq i < j \leq |S|$. We denote the i -th symbol of S by $S[i]$ and $S[i..j]$ is the subsequence $S_i S_{i+1} \dots S_j$.

We distinguish crossing and unlimited sets of arcs. A set P , where each sequence position is involved in at most one arc, i.e. $\forall (i, j) \neq (i', j') \in P : i \neq i' \wedge j \neq j' \wedge i \neq j \wedge i' \neq j'$, is called *crossing*. Otherwise it is called *unlimited*.

An *alignment* A of two arc-annotated sequences (S_a, P_a) and (S_b, P_b) is a ordered partial matching between the positions of S_a and S_b . More precisely, $A \subseteq \{1, \dots, |S_a|\} \times \{1, \dots, |S_b|\}$ has to satisfy for all $(i, i'), (j, j') \in A$ that 1.) $i > j$ implies $i' > j'$ and 2.) $i = j$ if and only if $i' = j'$. We define the (i, i') -*prefix* of A as $A \cap \{(j, j') \mid j \leq i, j' \leq i'\}$ and the (i, i') -*suffix* of A as $A \cap \{(j, j') \mid j > i, j' > i'\}$.

Fix two arc-annotated sequences (S_a, P_a) and (S_b, P_b) with unlimited structures P_a and P_b . Define the *weight of alignment* A of (S_a, P_a) and (S_b, P_b) as

$$\text{weight}(A) := \sum_{(i, i') \in A} \sigma(i, i') + \sum_{\substack{(i, j) \in P_a, (i', j') \in P_b, \\ (i, i') \in A, (j, j') \in A}} \tau(i, j, i', j') + \gamma(|S_a| + |S_b| - 2|A|),$$

where $\sigma(i, i')$ is the similarity of sequence positions $S_a[i]$ and $S_b[i']$, $\tau(i, j, i', j')$ is the similarity of arcs $(i, j) \in P_a$ and $(i', j') \in P_b$ and γ is the gap cost associated with each sequence position that is not matched.

The *alignment problem* is to determine

$$\operatorname{argmax}_{A \text{ alignment of } (S_a, P_a) \text{ and } (S_b, P_b)} \text{weight}(A).$$

Note that on crossing arc annotation, the ILP approach Lara [11] solves essentially the same problem. On unlimited input, Lara scores only a crossing subset of the matched arcs whereas our approach scores all matches of arcs.

2 Constraint Model

We model an alignment of arc-annotated sequences (S_a, P_a) and (S_b, P_b) by variables MD_i and M_i for $1 \leq i \leq |S_a|$ with initial domains $D(MD_i) = \{1, \dots, |S_b|\}$ and $D(M_i) = \{0, 1\}$. We write \mathbf{MD} and \mathbf{M} to denote the vectors of respective variables MD_i and M_i .

A valuation V of these variables corresponds to at most one alignment A_V of (S_a, P_a) and (S_b, P_b) as defined by

$$\begin{aligned} V(MD_i) = j \wedge V(M_i) = 1 &\text{ iff } (i, j) \in A \\ V(MD_i) = j \wedge V(M_i) = 0 &\text{ iff } \nexists j \text{ with } (i, j) \in A \\ &\wedge \forall (i', j') \in A : i' < i \rightarrow j' \leq j \wedge i' > i \rightarrow j' > j. \end{aligned}$$

In this way, M_i tells whether i is matched or deleted and the value j of MD_i tells that i is matched to j or deleted after j . One can show that for each alignment A of (S_a, P_a) and (S_b, P_b) there is a corresponding valuation V with $A = A_V$.

For example, the following alignment $A = \{(1, 1), (2, 4), (4, 5)\}$ of $S_a = ACUG$ and $S_b = ACACG$, which is often written as $\begin{matrix} A--CUG \\ ACAC-G \end{matrix}$, corresponds to the valuation $\mathbf{MD} = (1, 4, 4, 5)$ and $\mathbf{M} = (1, 1, 0, 1)$.

We introduce a constraint $\text{StringAlignment}(\mathbf{MD}, \mathbf{M})$ that is satisfied by any valuation with a corresponding alignment. For modeling the weight of the alignment, we introduce a variable Weight and a constraint $\text{StringAlignmentWeight}(\mathbf{MD}, \mathbf{M}, \text{Weight})$. This constraint relates a valuation of \mathbf{MD} and \mathbf{M} to the weight of its corresponding alignment. Note that, formally, (S_a, P_a) and (S_b, P_b) are parameters of the constraints but we omit them to simplify notation.

Both constraints are propagated by our propagator described in the next section. For finding optimal alignments we perform a branch-and-bound search enumerating \mathbf{MD} and \mathbf{M} according to a specific search strategy described after introducing the propagator itself.

3 The Alignment Propagator

Our propagator computes hyper-arc consistency for $\text{StringAlignment}(\mathbf{MD}, \mathbf{M})$ and propagates $\text{StringAlignmentWeight}(\mathbf{MD}, \mathbf{M}, \text{Weight})$.

It prunes \mathbf{MD} and \mathbf{M} due to the weight by computing upper bounds of weights for single variable assignments and furthermore computes lower and upper bounds for Weight based on \mathbf{MD} and \mathbf{M} . Computing such bounds efficiently is essential for branch-and-bound optimization.

Define the class $\mathcal{A}(D)$ as union of A_V over all valuations V that satisfy D . The computation of bounds is based on a relaxation of the alignment problem.

In this relaxation the two ends of each arc match are decoupled. Thus in the *relaxed optimization problem for D*, we maximize a relaxed weight

$$\text{weight}_{\text{relaxed}}^{n,m}(A) := \sum_{(i,i') \in A} [\sigma(i,i') + \text{ub}_D(i,i')] + \gamma(n+m - |A|),$$

over all alignments in $\mathcal{A}(D)$, where $n = |S_a|$ and $m = |S_b|$ and

$$\text{ub}_D(i,i') := \frac{1}{2} \max_{A \in \mathcal{A}(D)} \left[\sum_{\substack{(i,j) \in P_a, (i',j') \in P_b, \\ (i,i') \in A, (j,j') \in A}} \tau(i,j,i',j') + \sum_{\substack{(j,i) \in P_a, (j',i') \in P_b, \\ (i,i') \in A, (j,j') \in A}} \tau(j,i,j',i') \right].$$

Here, ub_D works as an upper bound for the weight contributions by arc matches involving (i,i') and consequently $\text{weight}_{\text{relaxed}}^{|S_a|,|S_b|}(A) \geq \text{weight}(A)$ for $A \in \mathcal{A}(D)$. Thus, solving the relaxed problem yields an upper bound of **Weight**.

For a moment, postpone how to efficiently compute $\text{ub}_D(i,i')$. Then, because the relaxed weight has the form of a sequence similarity score, one can apply the Smith-Waterman algorithm [10] to maximize the relaxed weight in $O(n^2)$ by dynamic programming, where $n = \max(|S_a|, |S_b|)$. The optimization problem is easily constrained due to domain D , because domains directly restrict the valid cases in the dynamic programming recursion.

Tracing back through the dynamic programming matrix yields an alignment A^l . If A_l also satisfies all other constraints of the constraint problem, then $\text{weight}(A^l)$ is a lower bound of **Weight**. For the later studied RNA alignment problem, this bound can always be propagated, since there are no other constraints. Furthermore, we compute upper bounds for each single variable valuation. This requires to complement the above “forward algorithm” that computes the matrix entries

$$\text{Prefix}(i,i') := \max_{(i,i')\text{-prefix } A_{ii'}^p \text{ of } A \in \mathcal{A}(D)} \text{weight}_{\text{relaxed}}^{i,i'}(A_{ii'}^p)$$

by a symmetric “backward algorithm” that computes the matrix entries

$$\text{Suffix}(i,i') := \max_{(i,i')\text{-suffix } A_{ii'}^s \text{ of } A \in \mathcal{A}(D)} \text{weight}_{\text{relaxed}}^{|S_a|-i, |S_b|-i'}(A_{ii'}^s).$$

Now the variables \mathbf{MD} can be pruned efficiently, because $\text{Prefix}(i,i') + \text{Suffix}(i,i')$ is an upper bound of **Weight** for the assignment $\text{MD}_i = j$. Hence, j can be removed from the domain of MD_i , if $\text{Prefix}(i,i') + \text{Suffix}(i,i')$ is larger than the upper bound of **Weight**. Similarly, we prune \mathbf{M} using the two matrices.

It remains to describe the efficient computation of $\text{ub}_D(i,i')$. It suffices to describe the maximization of $\sum_{\substack{(i,j) \in P_a, (i',j') \in P_b, \\ (i,i') \in A, (j,j') \in A}} \tau(i,j,i',j')$ over alignments in

$\mathcal{A}(D)$. A single match (j,j') can occur in an alignment in $\mathcal{A}(D)$ if $j' \in D(\text{MD}_j)$ and $1 \in D(\mathbf{M}_j)$. However, we look for the best set of simultaneously valid matches (j,j') . The structure of this subproblem is analogous to sequence alignment. For solving it efficiently, we define sorted lists j_1, \dots, j_l and j'_1, \dots, j'_l such that

$(j_h, i) \in P_a$ for all $1 \leq h \leq l$ and $(j_{h'}, i') \in P_b$ for all $1 \leq h' \leq l'$. We apply dynamic programming for evaluating

$$\begin{aligned} \text{UL}(0, 0) &= 0 & \text{UL}(h, 0) &= 0 & \text{UL}(0, h') &= 0 \\ \text{UL}(h, h') &= \\ \max \begin{cases} \text{UL}(h-1, h') & \text{unless } \text{MD}_{j_h} = j_{h'} \text{ and } \text{M}_{j_h} = 1 \\ \text{UL}(h, h'-1) & \text{unless } \text{MD}_{j_h} = j_{h'} \text{ and } \text{M}_{j_h} = 1 \\ \text{UL}(h-1, h'-1) + \tau(j_h, j_{h'}, i, i') & j_{h'} \in D(\text{MD}_{j_h}) \text{ and } 1 \in D(\text{M}_{j_h}) \end{cases} \end{aligned}$$

for $1 \leq h \leq l, 1 \leq h' \leq l'$. Then, we perform the same construction for the respective r and r' many arcs to the right of i and i' and evaluate the corresponding recursion equation $\text{UR}(h, h')$ for $1 \leq h \leq r, 1 \leq h' \leq r'$. Then, $\text{ub}_D(i, i') = \text{UL}(l, l') + \text{UL}(r, r')$.

Therefore, $\text{ub}_D(i, i')$ is computed in $O(ll' + rr')$ time. For crossing arcs $l+r = l' + r' = 1$ and for many other applications $l+r$ and $l' + r'$ can be constantly bounded [12] such that the propagator runs in $O(n^2)$ time and space.

Affine Gap Cost. In bioinformatics, penalizing unmatched positions using an affine weighting function yields more realistic results. Our method is straightforwardly extended to such scoring by using a Gotoh-like forward and backward algorithm [4] in the propagator without increasing its complexity. It appears that this modification comes more natural in our approach than the corresponding extension in ILP, because it does not require any change of the model.

Propagator-guided Search Strategy. To maximize the use of the propagator in a branch-and-bound setting, we suggest a search strategy that aims at disproving overestimated bounds fast and finding valid good alignments quickly. To achieve this, information computed by the propagator in each propagation step can be used to guide the search. In particular, this allows to select a variable that yields a high undecided contribution to the upper bound of the total weight. Furthermore, the computed backtrace provides a good candidate for a solution, which can be favored by the search strategy.

In our application to RNA, we select a variable with highest undecided contribution to the upper bound and domain size as tie breaking. Its domain is split such that the 20% highest relaxed weights are chosen first.

4 Problem Decomposition

Certain constraint optimization problems can be solved faster by detecting independent subproblems during search and optimizing these subproblems independently. In our setting, independent means that the two parts of the alignment do not depend on each other due to the string order, arcs connecting the two parts, or due to additional constraints other than `StringAlignment` and `StringAlignmentWeight`. In the following, we denote anything that makes two subproblems dependent a dependency, not only dependencies introduced by arcs.

In general, problem decomposition introduces overhead for detecting dependencies and even interferes with Branch-and-Bound when subproblems cannot be bounded well (confer AND/OR search [7], which however doesn't discuss decomposition in the presence of global propagators). However, the string alignment problem suggests a special form of decomposition along the string order, where our propagator provides upper bounds for the partial problems. The dependency due to the string order between the two subproblems for variables $MD_1, \dots, MD_{i-1}, M_1, \dots, M_{i-1}$ and $MD_{i+1}, \dots, MD_{|S_a|}, M_{i+1}, \dots, M_{|S_a|}$ is resolved as soon as a matching edge (i, j) is assigned (i.e. $MD_i = j$ and $M_i = 1$). The problem can be decomposed, if in addition the dependencies due to arcs are resolved and the corresponding variables do not have dependencies by other constraints. Notably, for resolving the dependencies due to an arc (i, j) it suffices that one of its ends is matched, i.e. $\forall A \in D(\mathcal{A}) : (i, i') \in A$ (or $\forall A \in D(\mathcal{A}) : (j, j') \in A$). Then one can move the weight of each arc match, $\tau(i, j, i', j')$, to the match weight of the other end $\sigma(j, j')$ (or $\sigma(i, i')$ respectively) and discard the dependency.

In the RNA alignment problem, all dependencies can be checked within the propagator [4]. To avoid overhead, we apply the decomposition only if an independent subproblem for $MD_i, \dots, MD_j, M_i, \dots, M_j$ can be solved to optimality by the propagator alone. This is the case, if all arc dependencies in the subproblem are resolved. Then, the problem reduces to maximum weighted string alignment without dependencies and the traceback alignment represents an optimal solution. Assigning the traceback of the subproblem to $MD_i, \dots, MD_j, M_i, \dots, M_j$ is a form of symmetry breaking, because it discards alignments with less or equal weight that are systematically generated from other alignments in $\mathcal{A}(D)$.

5 Results

The propagator and its application to RNA sequence-structure alignment, called Carna, is implemented in C++ using the constraint programming system Gecode. For handling input and output as well as for special data structures we reused code of LocARNA [12].

We run tests for two application scenarios. All experiments were performed under 32-bit Linux on a T400s notebook with Intel P9600 CPU. First, we explore Carna's behavior on crossing input structure using instances from all 16 Rfam families with crossing structure. Table 1 compares our results to Lara [1]. The table omits all 8 instances where both approaches run in less than 0.1 seconds. In all but one of the omitted cases, Carna solves the problem without backtracking. In terms of performance, with the single exception of tmRNA, both programs are on a par for the simpler class of crossing structures.

¹ This is important for our implementation, because checking for independent subproblems in the presence of arbitrary propagators is expensive in Gecode.

Table 1. Results for the eight harder instances of the benchmark set with crossing structures. We omit details for 8 instances where both programs run in less than 0.1 seconds. All times are given as user times.

Family	Lengths		Run-time (s)		Carna Search Tree		
	S_a	S_b	Carna	Lara	Depth	Fails	Size
Entero_OriR	126	130	0.03	0.18	38	13	50
Intron_gpI	443	436	0.1	0.2	0	0	1
IRES_Cripavirus	202	199	0.2	0.04	157	127	296
RNaseP_arch	303	367	0.46	1.4	63	8	64
RNaseP_bact_b	408	401	3.0	2.3	370	677	1463
RNaseP_nuc	317	346	0.07	2.9	14	4	16
Telomerase-vert	448	451	0.47	2.3	146	32	161
tmRNA	384	367	63	3.7	433	14347	28785

Table 2. Rfam Riboswitches. For 100 instances from each Rfam family annotated as riboswitch and confirmed by literature: Medians of average sequence length, average number of base pairs, and run-time (user time), maximal memory requirement and percentage of instances not solved to optimality within a given time limit of 1 min.

Family	Length	Base pairs	Time (s)	Memory (MB)	Limit
SAH_riboswitch	79	81	0.13	3.3	2%
SAM_alpha	79	96	0.03	1.9	0%
Purine	101	74	0.07	2.3	0%
Glycine	101	83	0.44	5.0	3%
SAM	106	74	0.06	6.0	0%
TPP	107	96	0.43	8.7	13%
SAM-IV	116	128	0.05	3.7	2%
MOCO_RNA_motif	141	111	0.24	9.4	10%
Lysine	181	210	60	14.5	60%
Cobalamin	204	237	60	18.7	71%

In our second scenario, we evaluate the behavior on the general class of unlimited structures. Therefore, we apply the approach to the alignment of riboswitches, which are RNA molecules with more than one evolutionary conserved structure. We annotate the RNA sequences by the set of all base pairs with sufficiently high probability in the RNA’s structure ensemble [8]. This set approximates the overlay of the different riboswitch structures. In consequence, the alignment is optimized with respect to all these structures simultaneously. Since the set of base pairs is unlimited this application has not been possible for existing approaches, which at most score crossing structure. The Rfam database contains 10 RNA riboswitch families that are confirmed by literature. To benchmark the approach, we align 100 random instances from each of those families. Since in large scale studies an instance is rather skipped than spending much

time on it, we set a time limit of 1 minute for each instance. The results are given in Table 2. For this new problem, not all instances could be solved within our strict time limit. However, the results show that the approach handles all Riboswitch families sufficiently well for bioinformatics applications with only some limitations for the very longest sequences.

6 Discussion

We presented a propagator for the problem of weighted string alignment with arbitrary pairwise dependencies, which is also known as the alignment problem for arc-annotated sequences with unlimited structure. Whereas the problem itself is MAX-SNP-hard, our propagator allows for an effective constraint programming approach by efficiently solving relaxations of the problem. Furthermore, we proposed a search strategy that improves the benefit due to the propagator. Finally, we showed that the weighted string alignment problem can be decomposed into independent subproblems during search. This allows for a AND/OR-type optimization in the context of our global propagator.

To evaluate the applicability of our method in practice, we apply it to the alignment of RNA structures. While all previous approaches in this area are limited to score at most crossing subsets of the input structures, our approach is able to align unlimited structures. This is useful to align Riboswitches and other molecules with more than one conserved structure, because it allows considering all their potential structural conformations simultaneously.

Acknowledgments. This work is partially supported by DFG grants WI 3628/1-1 and BA 2168/3-1 and PRIN-2008 (*Innovative multi-disciplinary approaches for constraint and preference reasoning*).

References

1. Bauer, M., Klau, G.W., Reinert, K.: Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization. *BMC Bioinformatics* 8, 271 (2007)
2. Blin, G., Fertin, G., Rusu, I., Sinoquet, C.: Extending the hardness of RNA secondary structure comparison. In: Chen, B., Paterson, M., Zhang, G. (eds.) *ESCAPE 2007*. LNCS, vol. 4614, pp. 140–151. Springer, Heidelberg (2007)
3. Caprara, A., Lancia, G.: Structural alignment of large-size proteins via lagrangian relaxation. In: *Proceedings of the Sixth Annual International Conference on Computational Biology (RECOMB 2002)*, pp. 100–108. ACM Press, New York (2002)
4. Gotoh, O.: An improved algorithm for matching biological sequences. *Journal of Molecular Biology* 162, 705–708 (1982)
5. Hoeve, W.-J., Pesant, G., Rousseau, L.-M.: On global warming: Flow-based soft global constraints. *Journal of Heuristics* 12(4-5), 347–373 (2006)
6. Jiang, T., Lin, G., Ma, B., Zhang, K.: A general edit distance between RNA structures. *Journal of Computational Biology* 9(2), 371–388 (2002)

7. Marinescu, R., Dechter, R.: And/or branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.* 173(16-17), 1457–1491 (2009)
8. McCaskill, J.S.: The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers* 29(6-7), 1105–1119 (1990)
9. Möhl, M., Will, S., Backofen, R.: Lifting prediction to alignment of RNA pseudoknots. *Journal of Computational Biology* (2010) (accepted)
10. Smith, T.F., Waterman, M.S.: Comparison of biosequences. *Adv. Appl. Math.* 2, 482–489 (1981)
11. Trick, M.A.: A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals OR* 118(1-4), 73–84 (2003)
12. Will, S., Reiche, K., Hofacker, I.L., Stadler, P.F., Backofen, R.: Inferring non-coding RNA families and classes by means of genome-scale structure-based clustering. *PLOS Computational Biology* 3(4), e65 (2007)

Using Learnt Clauses in MAXSAT

Jessica Davies, Jeremy Cho, and Fahiem Bacchus

Department of Computer Science, University of Toronto, Canada
{fbacchus, jkcho, jdavies}@cs.toronto.edu

Abstract. MAXSAT is an optimization version of SAT capable of expressing a variety of practical problems. MAXSAT solvers have been designed to take advantage of many of the successful techniques of SAT solvers. However, the most important technique of modern SAT solvers, clause learning, has not been utilized since learnt clauses cannot be soundly added to a MAXSAT theory. In this paper we present a new method that allows SAT clause learning to be exploited in a MAXSAT solver without losing soundness. We present techniques for learning clauses during a branch and bound (B&B) MAXSAT search, a process that is more complicated than standard clause learning. To exploit these learnt clauses we develop a connection between them and bounds that can be used during B&B. This connection involves formulating a hitting set problem and finding bounds on its optimal solution. We present some new techniques for generating useful hitting set bounds and also show how linear and integer programs can be exploited for this purpose, opening the door for a hybrid approach to solving MAXSAT.

1 Introduction

MAXSAT, in its most basic form, is the problem of finding a truth assignment that satisfies the maximum number of clauses of a CNF theory. Partial-MAXSAT is the extension in which some clauses are **hard**. Here the objective is to find a truth assignment that satisfies all the hard clauses and a maximum number of the other (soft) clauses. In weighted MAXSAT, clauses are assigned weights, and the objective is to find a truth assignment that maximizes the sum of the weights of the satisfied clauses. Weighted partial MAXSAT extends weighted MAXSAT by adding hard clauses that must be satisfied.

MAXSAT plays a fundamental role for optimization problems, similar to the role that SAT plays for satisfiability problems. Any optimization problem over finite domain variables can be encoded in MAXSAT. Thus MAXSAT can encode any finite domain MAX-CSP problem [5] (where the aim is satisfy as many constraints as possible), while the weighted versions of MAXSAT can encode most VALUED-CSPS [17]. MAXSAT serves as a general modeling language for such problems.

MAXSAT also has the advantage that it possesses a different structure than the corresponding MAX-CSP and VALUED-CSP formalisms. Each formalism can yield algorithmic insights that can potentially be exploited in the other. See [9,6] for an illustration of this type of cross-fertilization.

The literature on exact MAXSAT solvers is fairly extensive and can be categorized into two main approaches. The first approach solves the problem by

solving a sequence of SAT problems, e.g., [13]. This approach can fully exploit modern SAT solver technology including clause learning, but has a number of drawbacks that we will discuss later. The second approach employs a depth-first branch and bound search, e.g., [7]. In this line of research the key contributions consist of various techniques for computing good lower bounds during the search. Most of these techniques can be understood as applying restricted forms of MAXSAT-resolution [10].

First it should be noted that clauses inferred via standard resolution cannot be soundly added to a MAXSAT theory. If Φ is a MAXSAT CNF theory, and c is a clause inferred by resolution, then it is not sound to add c to Φ . In particular, the new theory $\Phi \cup \{c\}$ can have different solutions than the original. For example, the truth assignment $\pi = (x = \text{true}, y = \text{false})$ is a solution to the MAXSAT theory $\Phi = \{(\neg x, y), (x), (\neg y)\}$: it falsifies only one clause. However, (y) can be inferred from Φ , but π is not a solution to $\Phi \cup \{(y)\}$: π falsifies two clauses while $(x = \text{true}, y = \text{true})$ falsifies only one.

In response to this, alternative rules of inference have been developed. In particular, the MAXSAT-resolution rule is an extension of ordinary resolution that is sound and complete for MAXSAT [3,9]. However, MAXSAT-resolution can generate a large number of additional “compensation” clauses with each inference. In fact, the MAXSAT-resolution of two clauses of length k and j can generate up to $k + j$ additional clauses. This makes it difficult to use MAXSAT-resolution during search. Hence, previous work has concentrated on finding restricted cases where MAXSAT-resolution type inferences can be more efficiently applied.

Furthermore, current solvers look for these restricted cases in the reduced MAXSAT theory. That is, at each node of the B&B search tree, the MAXSAT theory has been reduced by the prefix of assigned variables. Current B&B MAXSAT solvers examine the reduced theory to determine if any restricted applications of MAXSAT-resolution can be supported. For example, in the reduced theory two originally long clauses might now be reduced to the unit clauses (x) and $(\neg x)$. These two unit clauses can be MAXSAT-resolved to generate the empty clause, thereby increasing the lower bound. However, in the original theory this resolution step actually corresponds to MAXSAT-resolving the two original long clauses, potentially generating many additional clauses which cannot be stored efficiently. As a result, all of the inferences made must be undone on backtrack and recomputed from scratch at future nodes in the search tree.

This is in stark contrast with SAT clause learning where clauses learnt in one part of the search tree can be utilized anywhere else. In fact, it is this caching and reuse of previous work that makes clause learning so effective.

In this paper we show how standard SAT clause learning can in fact be utilized in any kind of MAXSAT problem (weighted and/or partial). This allows us to learn clauses without having to generate large numbers of additional clauses. Hence we can apply clause learning in many cases where an equivalent MAXSAT-resolution would be impractical. Furthermore, clauses learnt in one part of the search tree or during a preprocessing phase can be reused throughout the rest of the search tree. As pointed out above, learnt clauses cannot be soundly added to a MAXSAT

theory. So to use these learnt clauses soundly we exploit a known relationship between conflicts and hitting sets [16]. This allows us to formulate a hitting set problem from the learnt clauses whose minimal solution provides a lower bound for the B&B search.

To exploit this connection we address two additional problems. First we develop techniques for learning clauses that are more likely to increase the minimal hitting set solution. Second, since computing a minimal hitting set is itself an NP-Complete problem, we develop some tractable techniques for computing lower bounds on the exact solution. We develop two heuristics for this purpose, one of which improves on the heuristic given in [16]. We also can formulate the minimal hitting set problem as an integer program and use linear programming to provide a lower bound approximation. We show how this link to standard OR techniques yields an interesting hybrid approach to the MAXSAT problem.

From our experimental results, we show that this novel approach to computing lower bounds demonstrates potential, in particular it can often quickly prune the search space. On several MAXSAT benchmarks, we show that enough inference can be made to terminate search at the root, while other state-of-the-art solvers, such as MiniMaxSat [7] require search. Our approach does not currently achieve state-of-the-art performance, although we are pursuing a number of ideas aimed at improving its performance. Furthermore, much recent research has focused on ways of improving existing bounding techniques, e.g., by exploiting cyclic structures [11]. Our approach offers a completely new way of producing improved bounds through finding specialized ways of learning new clauses. Finally, we believe that the techniques we present for computing bounds on minimal hitting sets and our empirical results assessing the trade offs between these techniques may also be of independent interest for other applications that rely on computing hitting sets.

In the sequel we first provide some brief background, followed in Section 3 by our results on the connection between solving MAXSAT and the weighted minimal hitting set problem. Section 4 describes proposed methods of approximating the min-weight hitting set, while in Section 5 we consider how to generate learnt clauses to employ in our bounds. We present preliminary experimental results in Section 6 that encourage future work in this direction, as discussed in the conclusion.

2 Background

For simplicity we will formulate the MAXSAT problem in terms of computing the minimal value of an objective function, as opposed to a truth assignment. We have as input a set of propositional variables x_1, \dots, x_n and a set of clauses c_1, \dots, c_m over these variables each with an associated weight $wt(c_i)$. Let Π be the set of 2^n truth assignments to the variables x_i . For $\pi \in \Pi$ the function $\pi(c)$ has value 1 if π falsifies the clause c and 0 if it satisfies it. The MAXSAT problem is to compute the minimum sum of weights of the falsified clauses over all truth assignments to the variables: $\min_{\pi \in \Pi} (\sum_{i=1}^m wt(c_i)\pi(c_i))$. Thus the weight of the

satisfied clauses is being maximized. Hard clauses are modeled as clauses c with $wt(c) = \infty$ (or any sufficiently large number). In the case of unweighted MAXSAT, we set $wt(c) = 1$ for all soft clauses c .

Solution methods: It is possible to approximate the optimal solution using methods like local search. But here we are mainly interested in exact solution methods, of which there are two main approaches taken in the literature.

In the first approach, e.g., [13], MAXSAT is solved by solving a sequence of SAT problems. Each run of the SAT solver checks to see if the current theory is UNSAT. If so, the theory is modified so that one (and only one) of the subset of clauses that caused the UNSAT result can be “turned off.” This is accomplished by adding new “turn-off” variables to the culprit clauses along with extra clauses constraining the turn-off variables so that only one can be true. Then the SAT solver is run again on the new problem. If the SAT solver returns SAT after k such runs we know that k is the minimum number of clauses that must be falsified when satisfying the remaining clauses.

Although this approach works well on some problems it quickly becomes inefficient when more and more clauses have to be turned off, resulting in SAT problems which are larger and more difficult to solve. In particular, in the i th run many more clauses have been added to handle the turn-off variables, and the solver must search over many combinations of i clauses to turn off. Thus the approach is less effective on highly constrained problems. A more serious limitation however lies in its extension to weighted MAXSAT problems [11, 14]. On such problems we must step through all possible values the MAXSAT solution can take, i.e., all distinct sums of clause weights. This can produce an explosion in the number of separate SAT solving episodes required.

The second approach involves a depth-first B&B search, which we also use in our work. Typically, a local search method is used to first compute an approximate solution, whose weight is taken as an initial upper bound on the optimal solution. During search, variables are assigned and any hard clauses (or clauses whose weight exceeds the upper bound) are used to perform unit propagation. After unit propagation various methods are used to compute a lower bound on the weight of clauses that must be falsified below the current node. This lower bound includes the weight of all clauses already falsified by the current assignment. In addition, inference is performed on the not-yet falsified (or satisfied) clauses to determine additions to the lower bound. For example, if two clauses have been reduced to units (x) and $(\neg x)$ then at every node below, one of these clauses will be falsified. Hence, $\min(wt((x)), wt((\neg x)))$ can be added to the lower bound. More powerful inference methods are used to handle more complex cases [7, 11]. However, as mentioned above, all of these inferences must be undone on backtrack.

3 Learnt Clauses—The Hitting Set Connection

We have already pointed out that learnt clauses cannot be soundly added to a MAXSAT theory (except if these clauses are inferred solely from the hard clauses of the theory). Nevertheless, they can be utilized in a MAXSAT solver.

First consider resolution proofs from a CNF MAXSAT theory Φ . Each proof p of a clause c_k is a sequence of clauses c_1, \dots, c_k , where each c_i is either an input clause from Φ or the result of resolving two previous clauses in the sequence. For any proof p let $ic(p)$ be the set of input clauses used in p , i.e., $p \cap \Phi$. For any set of proofs P let $ic(P)$ be the **set of sets** of input clauses arising from these proofs $\{ic(p) | p \in P\}$.

Initially we will be interested in **refutations**. These are proofs of the empty clause, i.e., $c_k = ()$. Since resolution is sound, any truth assignment that satisfies the input clauses of a refutation must also satisfy $()$. However, no truth assignment can satisfy $()$, hence no truth assignment can satisfy all of these clauses. That is, the input clauses of each refutation form a conflict set [16]. Since we are considering conflicts detectable by a complete inference technique (resolution) we can prove a more general connection between conflicts and solutions to MAXSAT than that given in [16].

For any MAXSAT CNF problem Φ let $minval(\Phi)$ be the solution of Φ , i.e., the minimum achievable weight of falsified clauses; and let $\mathcal{R}_{all}(\Phi)$ be the set of **all refutations** that can be computed from Φ . A hitting set for any **set of sets** of clauses S is a set of clauses \mathcal{H} such that for each $s \in S$, $\mathcal{H} \cap s \neq \emptyset$. The weight of a hitting set \mathcal{H} , $val(\mathcal{H})$ is the sum of the weights of the clauses in \mathcal{H} , and we let $minwt_{HS}(S)$ denote the weight of a minimum weight hitting set of S . Hence, $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi)))$ is the weight of a minimum weight hitting set of $ic(\mathcal{R}_{all}(\Phi))$.

Theorem 1. *For any MAXSAT theory Φ , $minval(\Phi) = minwt_{HS}(ic(\mathcal{R}_{all}(\Phi)))$.*

Proof. Let \mathcal{H} be a minimum weight hitting set of $ic(\mathcal{R}_{all}(\Phi))$.

We cannot prove the empty clause from $\Phi - \mathcal{H}$, otherwise there would be a refutation r in $\mathcal{R}_{all}(\Phi)$ such that $ic(r) \subseteq (\Phi - \mathcal{H})$. But then \mathcal{H} would not hit $ic(r)$ and \mathcal{H} would not be a hitting set of $ic(\mathcal{R}_{all}(\Phi))$. Thus we conclude that $minval(\Phi) \leq val(\mathcal{H})$. Say that $minval(\Phi) < val(\mathcal{H})$. Then there must exist a set of input clauses H with $val(H) < val(\mathcal{H})$ and such that $\Phi - H$ is satisfiable. But since \mathcal{H} is a minimum weight hitting set, H cannot be a hitting set of $ic(\mathcal{R}_{all}(\Phi))$. Hence, there exists a refutation $r \in \mathcal{R}_{all}(\Phi)$ such that $H \cap ic(r) = \emptyset$. That is, r is a refutation provable from $\Phi - H$. This is a contradiction and we conclude that $minval(\Phi) = val(\mathcal{H})$. □

Theorem 1 says that the technique of finding hitting sets can solve the MAXSAT problem: it is a complete method. However, as stated it is also quite impractical. For one there are an exponential number of possible refutations of Φ . Consider the case where instead of having access to $\mathcal{R}_{all}(\Phi)$ we have an incomplete collection of refutations $R \subset \mathcal{R}_{all}(\Phi)$. In this case a minimum hitting set of $ic(R)$ provides a lower bound on $minval(\Phi)$.

Proposition 1. *If R and R' are two sets of proofs with $R \subset R'$, then $minwt_{HS}(ic(R)) \leq minwt_{HS}(ic(R'))$.*

¹ The NP-Hardness of computing a minimum hitting set is also a problem with the direct application of Theorem 1. We address this issue in the next section.

This holds since every hitting set of $ic(R')$ must be a hitting set of $ic(R)$. Thus we have that $minwt_{HS}(ic(R)) \leq minwt_{HS}(ic(\mathcal{R}_{all}(\Phi))) = minval(\Phi)$.

Now we consider how these ideas could be utilized inside of a B&B search. At each node n of the search the original problem has been reduced by instantiating some set of variables. In addition, by employing clause learning techniques (described below) the search has been able to augment the input clauses Φ with an additional set of learnt clauses \mathcal{L} . For each $\ell \in \mathcal{L}$ the search has also computed the set of input clauses $ic(\ell)$ used in the proof of ℓ . We note that if c is an input clause ($c \in \Phi$) then a proof of c is simply c itself. Thus for convenience for $c \in \Phi$ we use $ic(c)$ to denote the set $\{c\}$.

At node n we wish to determine if $minval(\Phi|_n)$ (the input formula reduced by the literals made true at node n) has a value that is so high that it precludes finding a better solution under n . Theorem [1](#) says that $minval(\Phi|_n) = minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n)))$ and we can lower bound this value using any subset of $ic(\mathcal{R}_{all}(\Phi|_n))$. Unfortunately we do not know any non-empty subset. The search has never visited n before, and thus has not derived any refutations from $\Phi|_n$.

Instead we will consider clauses that we know to be falsified at node n . Let Π_n be the set of all truth assignments (to the variables in Φ) that agree with the assignments made at node n . Let $minval(n) = \min_{\pi \in \Pi_n} (\sum_{i=1}^m wt(c_i)\pi(c_i))$; this is the minimum weight of falsified clauses achievable by any truth assignment that lies below node n . We can backtrack immediately from n if we know any lower bound LB such that $LB \leq minval(n)$ and $LB \geq UB$ where UB is the value of the current best known solution.

Let $\mathcal{P}_{all}(\Phi, n)$ be the set of all proofs from Φ that derive a clause falsified by the assignments made at node n . That is, $p \in \mathcal{P}_{all}(\Phi, n)$ means that p derives, from the clauses of Φ , a clause all of whose literals are falsified at node n .

Theorem 2. $minval(n) = minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$.

This means that if we had access to all proofs of clauses falsified at node n , we could find $minval(n)$ by way of a minimum hitting set problem. We do not have access to $\mathcal{P}_{all}(\Phi, n)$, but as noted above for every clause c of $\Phi \cup \mathcal{L}$ that has been falsified at n we know $ic(c)$, the input clauses used in the proof of c . Hence, $FC = \{ic(c) \mid c \text{ is falsified at } n\} \subset ic(\mathcal{P}_{all}(\Phi, n))$ and Prop. [1](#) tells us that $minwt_{HS}(FC) \leq minval(n)$. By computing $minwt_{HS}(FC)$ the search can use this value to potentially backtrack away from n .

Proof. The proof of this theorem is more complicated than the proof of Theorem [1](#), so we provide only a sketch. First, let $\Phi|_n$ be Φ reduced by the assignments made at n , i.e., all satisfied clauses and falsified literals are removed from Φ . It can be shown that $minval(\Phi|_n) = minval(n)$ by observing that for every clause of Φ falsified at n , $\Phi|_n$ contains an empty clause of equal weight. Theorem [1](#) then shows that $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) = minval(n)$. Now all that remains to be done is to prove that $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) = minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$. First, we show that all proofs in $\mathcal{R}_{all}(\Phi|_n)$ can be converted to equivalent proofs in $\mathcal{P}_{all}(\Phi, n)$ by adding back all of the falsified literals to the clauses of the proof. This shows that any hitting set of $ic(\mathcal{P}_{all}(\Phi, n))$ can generate a hitting set of

$ic(\mathcal{R}_{all}(\Phi|_n))$ of no greater weight, and thus that $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) \leq minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$. The other direction is more complex, but we use the same argument as Theorem [II](#) to show that if $minwt_{HS}(ic(\mathcal{R}_{all}(\Phi|_n))) < minwt_{HS}(ic(\mathcal{P}_{all}(\Phi, n)))$ there must be a hitting set H of $ic(\mathcal{R}_{all}(\Phi|_n))$ which when converted to a set of clauses of Φ (by adding back the falsified literals) cannot be a hitting set of $ic(\mathcal{P}_{all}(\Phi, n))$. Hence, there is a proof $p \in \mathcal{P}_{all}(\Phi, n)$ not covered by H . With a more complex transformation, p can then be converted into a refutation in $\mathcal{R}_{all}(\Phi|_n)$ by removing all satisfied clauses and falsified literals, and then fixing all of the now broken resolution steps. (For example, the literal being resolved on might have been removed from one of the clauses, or one of the clauses might have been satisfied). The conversion of p is not hit by H contradicting that H exists. \square

In summary, the results of this section show that learnt clauses can be exploited in MAXSAT to produce lower bounds. In particular, at any node n of the search tree some set of clauses will be falsified. These could be either input or learnt clauses. By keeping track of the input clauses used to derive each clause a hitting set problem can be set up. The minimum weight hitting set provides a lower bound on the best value that can be achieved below node n , $minval(n)$, which can be used by the B&B bounding procedure. Furthermore, it can be the case that some learnt clauses are falsified at node n even though no input clause is violated. Hence, clause learning can allow us to construct richer hitting set problems that can yield better bounds.

Two problems remain. First we cannot necessarily compute a minimum weight hitting set as this is an NP-Complete problem in itself. In the next section we present some ways of computing lower bounds on the minimum weight hitting set, which in turn act as lower bounds on $minval(n)$. Second, we present some ways that clauses can be learnt both prior to search and during search so as to increase the effectiveness of the computed lower bounds.

4 Lower Bounding the Minimal Hitting Set

During clause learning we remember for each learnt clause the set of input clauses used in its derivation. This increases the space required to store learnt clauses, but does not impose additional computational overheads during search. In particular, we only need to access this derivation set when a clause has been falsified.

During search, we use standard watched literal techniques to detect when clauses become false. The collection of falsified clauses at each node of the search tree form a hitting set problem. Let FC represent this collection. For each $f \in FC$ let $ic(f)$ be the set of input clauses used to derive it, and $ic(FC) = \bigcup_{f \in FC} ic(f)$. As noted above, if f is an input clause, i.e., $f \in \Phi$, we let $ic(f) = \{f\}$. The hitting set problem is to select a minimum weight set of input clauses that touches $ic(f)$ for each $f \in FC$. Due to the difficulty of computing this, our aim is to compute a lower bound on the weight of a minimal weight hitting set.

Simplification: Before trying to solve the problem we first apply two quite effective simplification rules [20].

1. If $f_1, f_2 \in FC$ with $ic(f_1) \subseteq ic(f_2)$ remove f_2 from FC . Any hitting set that hits f_1 will necessarily hit f_2 .
2. If $wt(c_1) \leq wt(c_2)$ and $\{f|c_1 \in ic(f)\} \supseteq \{f|c_2 \in ic(f)\}$ remove c_2 from all sets $ic(f)$, $f \in FC$. We can substitute c_1 for c_2 in any hitting set without increasing the hitting set's weight.

Note that these two rules define a propagation scheme. That is, one application of these rules can enable additional applications. Simplification continues until neither of these rules can be applied again.

In addition to these two rules, which are applicable for any hitting set, we can also utilize an additional simplification rule specific to our problem.

3. Remove all input clauses c that are satisfied at the current node from $ic(f)$, $f \in FC$.

This last rule is justified by the semantics of our hitting sets. Each falsified clause implies that at least one of the input clauses used to derive it must be falsified by all truth assignments extending the current node. Thus, any clause that is already satisfied is no longer a candidate for falsification.

One thing that is useful to notice is that simplification often generates a collection of disjoint hitting set problems. For example, say that for falsified clause f we have $ic(f) = \{c\}$, perhaps because simplification removed all other members of $ic(f)$. Rule 1 then implies that all other falsified clauses f' with $c \in ic(f')$ will be removed from the hitting set problem. That is, an isolated hitting set problem consisting only of $f = \{c\}$ will be created. These disjoint problems can be solved independently and the answers added.

Heuristics: We have developed two heuristics for lower bounding the minimum weight hitting set.

H1 (LB)

1. LB = 0
2. **while** $FC \neq \emptyset$
3. **choose** $f \in FC$
4. LB += $\min_{c \in ic(f)} wt(c)$
5. $F = \{f' \in FC | ic(f) \cap ic(f') \neq \emptyset\}$
6. $FC = FC - F$

This heuristic first chooses some falsified clause, f , and adds its minimum weight input clause to the lower bound. Then it removes f and all clauses that share an input clause with f from the set of falsified clauses. It repeats this loop until no more falsified clauses remain.

The intuition behind this heuristic is simple: we can hit f by selecting its min-weight input clause. But we could have selected any other input clause from $ic(f)$. Hence, the most we could have done is also hit all other falsified clauses connected to f via a member of $ic(f)$. The heuristic conservatively estimates that we did in fact hit all of these clauses with f 's min-weight clause. Note

that the heuristic can yield different values dependent on which f is chosen. In our implementation, we use different selection schemes for f in the weighted and unweighted cases. For weighted, we always select the f with the minimum weight input clause of maximum weight, while for unweighted, we select the f whose set F in line 5 is of minimum cardinality. However, other natural selection schemes also exist.

This heuristic inherently takes advantage of any disjoint sub-problems. In particular, if the hitting set problem has been broken up into k disjoint sub-problems, H1 will return an LB that is no worse than the sum of the weights of the minimum weight input clause present in each subproblem.

For the second heuristic, for input clause c let $nbrs(c) = \{f | f \in FC \wedge c \in ic(f)\}$ and $deg(c) = |nbrs(c)|$, that is the number of falsified clauses it hits.

H2(LB)

1. LB = 0
2. **foreach** disjoint subproblem FC_P
3. lb = 0, n = $|FC_P|$
4. **while** n > 0
5. $c = c \in ic(FC_P)$ that minimizes $wt(c)/deg(c)$
6. **if** $deg(c) \geq n$ OR unweighted clauses
7. lb += $wt(c)$
8. **else** lb += $n \times wt(c)/deg(c)$
9. n -= $deg(c)$
10. remove c from $ic(f)$ for all $f \in FC_P$
11. LB += lb

This heuristic generalizes one given in [16]. First it takes advantage of the possible disjointness of the hitting set problem that might arise after simplification, and second it handles the case of weighted input clauses. It operates on each disjoint subproblem by selecting input clauses with lowest weight over degree. These clauses hit the most sets on a minimum cost per set basis. We select enough such minimum average cost input clauses until the sum of their degrees equals or exceeds the total number of sets to hit. However, in the case of weighted clauses, we are only allowed to count part of the weight of the last clause selected (line 8).

Proposition 2. *Both H1 and H2 return a lower bound on the weight of the minimum weight hitting set.*

This proposition is easy to see for H1 and for H2 in the unweighted case. The weighted case requires a bit more insight, but ultimately it also is not too difficult.

These two heuristics are incomparable. That is, on some problems H1 provides a better bound than H2 and vice versa on other problems.

For example, let $FC = \{f_1, f_2, f_3\}$, $ic(FC) = \{i_1, i_2, i_3\}$ (all unweighted), where $nbrs(i_1) = \{f_1, f_2\}$, $nbrs(i_2) = \{f_1, f_3\}$, $nbrs(i_3) = \{f_2, f_3\}$. Then H1 can only pick a single element from FC for a LB of 1 while H2 can pick any two elements from $ic(FC)$ for a LB of 2.

On the other hand, let $FC = \{f_1, f_2, f_3, f_4, f_5, f_6\}$, $ic(FC) = \{i_1, i_2, i_3\}$ (all unweighted), where $nbrs(i_1) = \{f_1, f_2, f_3\}$, $nbrs(i_2) = \{f_3, f_4, f_5\}$, and $nbrs(i_3) = \{f_5, f_6\}$. Then H1 can pick $\{f_1, f_4, f_6\}$ for a LB of 3 while H2 is forced to pick $\{i_1, i_2\}$ for a LB of 2.

Unfortunately both heuristics can yield arbitrarily bad approximations.

Theorem 3. *For hitting set instance \mathcal{S} , let $H1(\mathcal{S})$ and $H2(\mathcal{S})$ be the lower bounds computed by the heuristics and $minwt_{HS}(\mathcal{S})$ be the weight of the minimum weight hitting set for \mathcal{S} . Then for any $\epsilon > 0$, there exists $\mathcal{S}, \mathcal{S}'$ such that $H1(\mathcal{S})/minwt_{HS}(\mathcal{S}) \leq \epsilon$ and $H2(\mathcal{S}')/minwt_{HS}(\mathcal{S}') \leq \epsilon$*

4.1 Integer Programming Connection

Recall from Section 3 that the quality of the lower bound on the minimum hitting set dictates how soon we can backtrack from a non-optimal partial assignment. Furthermore, by recomputing the lower bound at each node during backtrack², it also impacts how far we can backtrack (as long as $LB \geq UB$). In light of this and Theorem 3, it may be desirable to use more powerful techniques to compute the exact value of the minimum hitting set at strategic points during search. One way to do this is to encode the hitting set instance as an integer program and solve it using a Mixed Integer Program (MIP) solver, such as CPLEX. A standard encoding [19] of a hitting set instance with $FC = \{f_1, \dots, f_n\}$, $ic(FC) = \{i_1, \dots, i_m\}$, where w_k is the weight of i_k , maps FC to constraints and $ic(FC)$ to boolean variables as follows:

$$\begin{aligned} \text{minimize: } & \sum_{k=1}^m w_k \cdot x_k & \text{where :} \\ & \text{for } 1 \leq j \leq n & \sum_{i_k \in ic(f_j)} x_k \geq 1 \\ & \text{for } 1 \leq k \leq m & x_k \in \{0, 1\} \end{aligned}$$

Because Integer Programming is itself NP-complete, the backtracking gains of an exact solution to the minimum hitting set problem may be outweighed by the overhead required to compute it, particularly for large instances. As a result, it can be more practical to instead solve the linear programming relaxation, whose solution is a valid lower bound on the minimum hitting set. To balance the trade off between the quality of the minimum hitting set approximation and the computational cost required, we employed the following strategy: first heuristics H1 and H2 are computed and their maximum used as initial lower bound LB. If $LB < UB$ but $LB/UB \geq \alpha$, for some tuned parameter α , then the linear program is solved. Finally, if this is still insufficient to exceed UB and if the size of the hitting set problem is less than some other tuned parameter β , the integer program is solved.

² Note that it may not be sufficient to simply reuse the lower bound that was computed the first time the node was reached, as additional learnt clauses may have been added to the hitting set instance since then.

5 Learning Clauses

It remains to consider how to generate the learnt clauses; we consider two methods. The first is to perform a preprocessing step to learn clauses during a relaxed DPLL search where soft as well as hard clauses are involved in learning [8]. A drawback to this approach is that the learnts may not be relevant to the subsequent B&B search. For one thing, all soft unit clauses are satisfied during relaxed DPLL but may have to be violated to find the optimum. As observed by Kroc et al., we found that some MAXSAT problems are too easy to refute even for relaxed DPLL, and in many cases few or no learnts can be generated this way. Therefore, we developed techniques to generate learnts during the B&B of a MAXSAT solver.

Soft Unit Propagation: At every node of the B&B search, unit propagation is applied over the hard (with respect to the current UB) clauses to soundly reduce the theory under the current prefix. If a hard conflict is found, we learn a hard clause and backtrack as normal to the asserting level. Otherwise, as in [7], we initialize a phase of *soft unit propagation* (SUP) with all soft clauses that are unit or falsified under the current prefix. Soft propagation involves unit propagating soft clauses as well as hard, so any literals set during the SUP phase must be undone before continuing search with another decision. However, if a clause is falsified during SUP, this conflict can be analyzed using standard techniques to produce a learnt with the desirable property of being falsified under the current prefix, *before* SUP. Thus it may contribute to increasing the hitting set lower bound at the current node. Therefore, the learnts produced are immediately relevant to the search, in contrast to those we obtained from the preprocessing step. The learnt clause, together with its set of input clauses, is saved upon backtrack and can be used in future search whenever it is falsified, to contribute to the hitting set lower bound. The soft learnts also participate in future SUP phases, so that soft learnts can be learned from others, contributing to the power of these clauses to prune the search.

SUP continues to propagate and learn until no more new falsified clauses are found. Thus many soft learnts can be produced at each node of the search, and we update the lower bound after SUP is finished. However, it could be beneficial to limit the amount of SUP learning performed at every node, or update the lower bound with new learnts as soon as we expect to be able to exceed the upper bound. The trade off between the strength of lower bound we can produce and time spent learning will be a focus of future investigation.

Turning off clauses: SUP learning, if implemented as described above, can produce duplicate learnts. Note that in our context, a learnt is only considered a duplicate of another if their sets of input clauses are also the same, since otherwise each can contribute to the lower bound under different circumstances depending on the structure of the hitting set problem. In order to prevent duplicate learnts from creating overhead, we prevent them from being learnt in the first place. This is achieved by “turning off” for SUP, one of the input clauses of each existing falsified learnt, for the duration of time it remains falsified. A

turned-off input clause can't be used to derive any more learnts, until it is turned back on. Of course, this policy may reject new learnts that aren't actually duplicates of an existing one. This is undesirable since it may reduce the strength of the lower bound. To limit the negative impact on heuristics H1 and H2, we always choose to turn off the min-weight input clause, or the one with largest degree. We also consider turning off input clauses only at decision levels greater than k and relying instead on a general scheme of learnt database reduction to limit the number of learnts.

Similarly, when an input clause c is falsified by the current prefix, we turn off all of the learnts it has derived. This prevents us from learning clauses that we know can't currently contribute to the lower bound. To see this, note that any learnt we avoid deriving would have had c in its set of input clauses. Therefore, while c is falsified, it would be used by the hitting set heuristics to cover all such learnts, so the weight of the hitting set would not be increased by their presence.

Related lower bounds: If no clauses are turned off for SUP, and the exact minimum weight hitting set is calculated for the lower bound, this technique subsumes existing unit-propagation based bounds in the MAXSAT literature [12,7,21]. If we turn off clauses for SUP, our lower bound will be at least as good as the disjoint-inconsistent sub-formulas bound [12], since the turned off input clause from one inconsistent sub-formula does not need to be used to refute the others.

6 Empirical Study

We implemented these ideas on top of the SAT solver Minisat [4], to investigate how they perform in practice. We incorporated the variable ordering heuristics used by MiniMaxSat, as well as the natural extension of probing (aka failed literal detection) to our clause learning framework [7]. We also use the Dominating Unit Clause rule to simplify the theory at each node [15]. Our experiments were conducted on a subset of 378 instances from the previous MaxSAT Evaluations [2]. The instances were selected by identifying benchmark families in which MiniMaxSat is unable to solve some instances, where our B&B solver can outperform MiniMaxSat on at least some problems. We then selected particular families in order to represent all types (MAXSAT, partial MAXSAT etc.) and some from each category (random, crafted, industrial). All experiments were conducted on a dual-core 2GHz AMD Opteron processor with 3GB of RAM, and all experiments were run with a 1200 second timeout.

H1 vs. H2: Our first set of experiments investigates the performance of the two hitting set approximation heuristics, H1 and H2, in the context of providing a lower bound during B&B search. We ran the B&B search with both heuristics enabled, and used the maximum of the two as the lower bound. We counted the number of times each of the two heuristics provided different bounds, and calculated the relative amount by which the winner was better. The results over the 226 instances that required at least 100 decisions to solve, are summarized in Table 1. The first column specifies the heuristic that was used, either H1

Table 1. Comparison of the H1 and H2 lower bounds during search. The ‘Freq’ column refers to the percentage of all lower bounds calculated for which the heuristic gave the larger bound, averaged over all instances. The ‘Size’ column gives the average factor by which the bound was larger. The other columns give the average number of decisions, average runtime, and number of instances solved.

LB Heuristic	Freq	Size	Decisions	Time (s)	Num Solved
H1	50	1.15	36280	49	115
H2	6	1.09	40115	50	115
max(H1,H2)	–	–	36192	49	117

Table 2. Comparison of the H1 and CPLEX LP lower bounds during search. The average number of decisions and runtime (over instances both methods solved), and the number of instances solved is shown.

LB Heuristic	Decisions	Time (s)	Num Solved
CPLEX LP	25296	48	105
H1	35059	15	115

Table 3. Comparison of MiniMaxSat and our B&B solver using the CPLEX ILP lower bound

Year	Type	Name	Optimum	MiniMaxSat Decisions	Our Decisions	MiniMaxSat Time (s)	Our Time (s)
2007	wpms	8.wcsp.log	2	1	0	0.05	0.05
2007	wpms	norm-mps-v2-20-10-stein15	9	2191970	0	3.8	0.07
2007	wpms	norm-mps-v2-20-10-stein27	18	–	0	>1200	0.59
2007	wpms	norm-mps-v2-20-10-stein9	5	230	0	0.12	0.14
2008	pms	norm-fir01_area_delay	5	14	0	0.14	0.12
2008	pms	norm-fir02_area_partials	19	38	0	0.16	0.06
2008	pms	norm-fir04_area_partials	30	13	0	0.12	0.6
2008	wms	frb10-6-1	50	755	0	0.27	0.23
2008	wms	frb10-6-2	50	678	0	0.13	0.26
2008	wms	frb10-6-3	50	1302	0	0.13	0.23
2008	wms	frb10-6-4	50	580	0	0.29	0.3
2008	wms	frb15-9-1	120	387470	0	1.37	3.76
2008	wms	frb15-9-2	120	206845	0	2.29	4.6
2008	wms	frb15-9-4	120	199365	0	2.24	5.77
2008	wms	frb15-9-5	120	271024	0	1.27	5.59
2009	wpms	warehouse0.wcsp	328	46	0	0.11	0.12

or H2 alone, or their maximum. The second column shows the percentage of all lower bounds calculated for which the one heuristic gave a larger bound than the other (averaged over all instances). Whenever one heuristic gave a strictly larger bound than the other, we measured the relative difference (e.g. H1/H2 if H1 was the larger); the third column reports this averaged over all instances. The average number of decisions and average runtime are shown in columns four and five, over the 112 instances that were solved by all three methods. The total number of instances solved using each method is included in the last column. These results encourage us to use both lower bounds and take the maximum, since they are both cheap to calculate and can solve more problems when combined.

H1 vs. CPLEX LP: We also consider the trade off between using our H1 heuristic, and solving the linear program for the hitting set problem using CPLEX.

We expected that the dynamic addition and removal of variables and constraints from the CPLEX model would limit the efficiency of this approach, and the results confirm that the added strength of the LP lower bound comes with the price of greater computational cost. We ran our B&B solver with the H1 lower bound alone, and then with only the CPLEX LP lower bound. Ninety-seven instances were solved by both methods, and the results are shown in Table 2. We see that by using the LP lower bound, we make 28% fewer decisions on average. We found that the reduction in decisions can sometimes pay off in reduced runtime, for 37 instances. However in the majority of cases, the LP bound increases the runtime by an average of about 30%. In general, the extra computational cost does not pay off, since using the stronger LP bound solves 10 fewer problems. These results confirmed our expectations, but demonstrate that a hybrid approach, using the stronger bounds at judicious points during search to exceed the UB, is a well-justified direction for future work.

Solving without search: Finally, we present some results that show the promising potential of our framework to allow stronger yet practical inference. As mentioned at the beginning of this section, we implemented a probing phase at the root of the B&B search. For each literal, we force it to *true* and reduce the theory with this assignment using first hard unit propagation, followed by SUP. If a clause is falsified, we learn a unit or empty clause (associated with a set of input clauses) and move on to probe the next literal. On some instances, the set of clauses we learn during probing is strong enough that the lower bound will equal a tight upper bound provided by one run of `ubcsat` [18]. This occurs on 16 problems, presented in Table 3. Note that we only report the cases where Mini-MaxSat couldn't solve the instance without search. Here, we have used CPLEX to solve the ILP model and generate the lower bound, since the size of the hitting set problem is sufficiently small.

7 Conclusions

We introduced an innovative approach for MAXSAT solving, with potential for practical impact based on generating bounds from unrestricted clause learning for MAXSAT. Although it may always be necessary to use a restricted version on real problems, we argue that this framework provides new insight into how strong lower bounds can be made practical, for example, by being smart about which soft clauses we learn, or by approximating the minimum hitting set well. In addition to these contributions, we present two heuristics for the weighted hitting set problem, and show that this approach can be used effectively in a novel context. Based on our preliminary implementation, we have discovered that the primary challenge in bringing this technique to the state-of-the-art in practical performance, will be to develop methods to learn the best clauses to prune the search tree. This is the topic of ongoing research.

Acknowledgment. This work was supported by the National Research Council of Canada.

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial maxsat through satisfiability testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009)
2. Argelich, J., Li, C.M., Manyà, F., Planes, J.: The maxsat evaluations (2007–2009)
3. Bonet, M.L., Levy, J., Manyà, F.: A complete calculus for max-SAT. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 240–251. Springer, Heidelberg (2006)
4. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
5. Freuder, E., Wallace, R.: Partial constraint satisfaction. *Artificial Intelligence (AI)* 58(1-3), 21–70 (1992)
6. de Givry, S., Larrosa, J., Meseguer, P., Schiex, T.: Solving max-SAT as weighted csp. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 363–376. Springer, Heidelberg (2003)
7. Heras, F., Larrosa, J., Oliveras, A.: MinimaxSAT: An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research (JAIR)* 31, 1–32 (2008)
8. Kroc, L., Sabharwal, A., Selman, B.: Relaxed dpll search for maxsat. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 447–452. Springer, Heidelberg (2009)
9. Larrosa, J., Heras, F.: Resolution in max-SAT and its relation to local consistency in weighted csps. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 193–198 (2005)
10. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-SAT solving. *Artificial Intelligence (AI)* 172(2-3), 204–233 (2008)
11. Li, C.M., Manyà, F., Mohamedou, N., Planes, J.: Exploiting cycle structures in max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 467–480. Springer, Heidelberg (2009)
12. Li, C.M., Manyà, F., Planes, J.: New inference rules for max-SAT. *Journal of Artificial Intelligence Research (JAIR)* 30, 321–359 (2007)
13. Liffiton, M., Sakallah, K.: Generalizing core-guided max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 481–494. Springer, Heidelberg (2009)
14. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009)
15. Niedermeier, R., Rossmanith, P.: New upper bounds for maximum satisfiability. *Journal of Algorithms* 36, 63–88 (2000)
16. Petit, T., Bessière, C., Régim, J.C.: A general conflict-set based framework for partial constraint satisfaction. In: *5th Workshop on Soft Constraints (Soft 2003)*, Kinsale, Ireland (2003)
17. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: Hard and easy problems. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 631–639 (1995)
18. Tompkins, D., Hoos, H.: Ubsat: An implementation and experimentation environment for sls algorithms for SAT and max-SAT. In: Hoos, H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 306–320. Springer, Heidelberg (2005)
19. Vazirani, V.: *Approximation algorithms*. Springer, Heidelberg (2001)
20. Weihe, K.: Covering trains by stations or the power of data reduction. In: *Proceedings of Algorithms and Experiments (ALEX 1998)*, pp. 1–8 (1998)
21. Xing, Z., Zhang, W.: Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence (AI)* 164, 47–80 (2005)

Domain Consistency with Forbidden Values

Yves Deville¹ and Pascal Van Hentenryck²

¹ Université catholique de Louvain
yves.deville@uclouvain.be

² Brown University
pvh@cs.brown.edu

Abstract. This paper presents a novel domain-consistency algorithm which does not maintain supports dynamically during propagation, but rather maintain forbidden values. It introduces the optimal NAC4 (negative AC4) algorithm based on this idea. It further shows that maintaining forbidden values dynamically allows the generic algorithm AC5 to achieve domain consistency in time $O(ed)$ for classes of constraints in which the number of supports is $O(d^2)$ but the number of forbidden values is $O(d)$. The paper also shows how forbidden values and supports can be used jointly to achieve domain consistency on logical combinations of constraints and to compute validity as well as entailment of constraints. Experimental results show the benefits of the joint exploitation of supports and forbidden values.

1 Introduction

In constraint programming, propagation aims at reducing the search space without removing solutions. The propagation algorithm considers each constraint individually and terminates when no constraint can be used to reduce the domains of the variables. The ideal propagation for a constraint is *domain consistency*, also known as arc consistency: It removes from the domain of each variable all values that do not belong to a solution of the considered constraint. Many algorithms have been proposed for achieving domain consistency, such as AC3, AC4, AC6, AC7 and AC2001 (see [1]). Consistency algorithms typically use the concept of *support*. For a binary constraint x over variables x and y , a support for a pair (x, a) , where a is a possible value for x , is a pair (y, b) such that $C(x/a, y/b)$ holds. The optimal time complexity to achieve domain consistency for a CSP is $O(e \cdot d^2)$ for binary constraints and $O(e \cdot r \cdot d^r)$ for non-binary constraints (d is the size of the largest domain, e the number of constraints and r the largest arity of the constraints). An algorithm such as AC4 maintains all the supports for all pairs (x, a) , while other algorithms (e.g., AC6) only maintain a single support and search for subsequent supports on demand. AC4 works in two steps. First, it computes all the supports for all the variable/value pairs in each constraint. Then, it propagates the removal of a value a from the domain of a variable x . An interesting property of the propagation step is that its time complexity is proportional to the total number of supports.

Example 1. Consider the constraint $x = y \bmod 10$, with $D(x) = \{0..9\}$ and $D(y) = \{0..99\}$, the size of the supports for x is linear ($O(\#D(y))$, where $\#A$ is the size of A). The propagation step of AC4 for this constraint is also linear, while it remains quadratic for other optimal AC algorithms such as AC6, AC7, or AC2001.

Of course, the initialization step of AC4, which computes all the supports, is $O(d^2)$ even if the number of supports is $O(d)$, since the algorithm has no knowledge of the semantics of the constraint. The generic AC5 algorithm [2] was designed to exploit the semantics of the constraints, and can then be used to generate the supports of such constraints in linear time, resulting in an $O(ed)$ complexity.

The scientific question addressed in this paper is the following: *Is it possible to design a domain-consistency algorithm running in time $O(ed)$ if the number of supports is quadratic, but the number of forbidden values (also called conflict set) is linear?*

Example 2. Consider the constraint $x \neq y \bmod 10$, with $D(x) = \{0..9\}$ and $D(y) = \{0..99\}$. The size of the supports for x is 900, hence a complexity of $O(\#D(x) \cdot \#D(y))$ in the propagation step of AC4. Using AC3, AC7 or AC2001 does not help to reduce this complexity. However, the size of the forbidden values is 100 ($O(\#D(y))$). Can we use an AC4-like algorithm that maintains the list of forbidden values instead of the supports to obtain an $O(ed)$ algorithm?

This paper answers this question positively and makes the following contributions:

- It proposes the NAC4 algorithm (Negative AC4) that achieves the optimal $O(e \cdot d^2)$ time complexity for binary CSPs, but dynamically maintains the set of forbidden values instead of supports. It shows that both AC4 and NAC4 are instances of the generic AC5 algorithm: they can be combined naturally in a single constraint solver and AC5 can exploit the constraint semantics to obtain higher efficiency. NAC4 is also generalized for non-binary CSPs.
- It identifies classes of constraints for which domain consistency can be achieved in linear time.
- It demonstrates how the combination AC4/NAC4 can achieve domain consistency on logical combinations of constraints over the same variables.
- It shows that the combination of AC4/NAC4 can easily be extended to provide methods assessing the validity and the entailment of a constraint.
- It presents experimental results showing the benefits of the combination AC4/NAC4.

Related Work. The idea of using forbidden values is not new in CP: what is novel in this paper is that NAC4 maintains the set of forbidden values dynamically during the propagation. References [3,4] use negative table constraints, where the table describes the set of forbidden tuples. The negative table is used to find the next support of a value by means of binary search; it is static however and not updated during propagation. Lecoutre [5] showed that (x, a) has a support for a constraint $c(x, y)$ if the size of $D(y)$ is strictly superior to the size of the initial conflict set of (x, a) . This idea is integrated in a coarse-grained algorithm. Once again, the size of the conflict set is not updated during the computation. The same idea is also proposed as a support condition in [6].

The consistency of a combination of constraints has been handled in different ways. Some approaches achieve domain consistency, which is NP-hard in general. A domain-consistency algorithm, based on AC7, was proposed in [7] for the conjunction of constraints. Lhomme [8] describes a domain-consistency algorithm for any combination of constraints. It focuses primarily on constraints given in extension. Forbidden tuples are used once again through a static negative table. Other approaches compute an

approximation of domain consistency, such as in [9] (cardinality), [10] (constructive disjunction), or [11] which provides an algebra for combining constraints.

2 AC5

This section revisits the generic AC5 algorithm [2], generalizing it slightly to accommodate AC4, AC6, and AC2001 as instantiations.

Definition 1 (CSP). A binary CSP $(X, D(X), C)$ is composed of a set of n variables $X = \{x_1, \dots, x_n\}$, a set of domains $D(X) = \{D(x_1), \dots, D(x_n)\}$ where $D(x)$ is the set of possible values for variable x , and a set of binary constraints $C = \{c_1, \dots, c_e\}$, with $\text{Vars}(c_i) \subseteq X$ ($1 \leq i \leq e$). We denote $d = \max_{1 \leq i \leq n} (\#D(x))$.

Let c be a constraint with $\text{Vars}(c) = \{x, y\}$, $a \in D(x)$, $b \in D(y)$. $c(x/a, y/b)$ or $c(y/b, x/a)$ denote the constraint where variables x and y have been replaced by the values a and b . We assume that testing $c(x/a, y/b)$ takes $O(1)$ time. If $c(x/a, y/b)$ holds, then $(x/a, y/b)$ is called a support of c and (y, b) is a support for (x, a) on c . If $c(x/a, y/b)$ does not hold, then $(x/a, y/b)$ is called a conflict of c and (y, b) is a conflict (or forbidden value) for (x, a) on c .

Definition 2. Let c be a constraint with $\text{Vars}(c) = \{x, y\}$. The set of inconsistent, consistent, and valid values for x in c wrt a set of values B are defined as follows:

$$\begin{aligned} \text{Inc}(c, x, B) &= \{(x, a) \mid a \in D(x) \wedge \forall b \in B : \neg c(x/a, y/b)\} \\ \text{Cons}(c, x, B) &= \{(x, a) \mid a \in D(x) \wedge \exists b \in B : c(x/a, y/b)\} \\ \text{Valid}(c, x, B) &= \{(x, a) \mid a \in D(x) \wedge \forall b \in B : c(x/a, y/b)\} \end{aligned}$$

We use $\text{Inc}(c, x)$ to denote $\text{Inc}(c, x, D(y))$ and similarly for the other sets.

Definition 3 (Domain Consistency). A constraint c over $\{x, y\}$ is domain-consistent wrt $D(X)$ iff $\text{Inc}(c, x) = \emptyset$ and $\text{Inc}(c, y) = \emptyset$. A CSP $(X, D(X), C)$ is domain-consistent iff all its constraints are domain-consistent wrt $D(X)$.

Specification [1] describes the principal methods used by AC5. The AC5 algorithm uses a queue Q of triplets (c, x, a) stating that the domain consistency of constraint c should be reconsidered because value a has been removed from $D(x)$. When a value is removed from a domain, the method `enqueue` puts the necessary information on the queue. In the postcondition, Q_o represents the value of Q at call time. The parameter $C1$ allows us to consider a subset of constraints, which will be necessary in the initialization. As long as (c, x, a) is in the queue, it is algorithmically desirable to consider that value a is still in $D(x)$ from the perspective of constraint c . This is captured by the following definition.

Definition 4. The local view of a domain $D(x)$ wrt a queue Q for a constraint c is defined as $D(x, Q, c) = D(x) \cup \{a \mid (c, x, a) \in Q\}$.

Example 3. Given a queue $Q = \{(c_1, y, 2), (c_1, z, 2), (c_2, y, 3)\}$ and domains $D(x) = \{1, 2\}$, $D(y) = D(z) = \{1\}$, then $D(x, Q, c_1) = D(y, Q, c_1) = D(z, Q, c_1) = \{1, 2\}$.

```

1 enqueue(in x: Variable;in a: Value;in C1: Set of Constraints;
2         inout Q: Queue)
3 // Pre:  $x \in X$ ,  $a \notin D(x)$  and  $C1 \subseteq C$ 
4 // Post:  $Q = Q_0 \cup \{(c, x, a) | c \in C1, x \in Vars(c)\}$ 
5
6 post(in c: Constraint;out  $\Delta$ : Set of Values)
7 // Pre:  $c \in C$  with  $Vars(c) = \{x, y\}$ 
8 // Post:  $\Delta = Inc(c, x) \cup Inc(c, y)$  + initialization of specific data structures
9
10 boolean valRemove(in c: Constraint;in y: Variable; in b: Value;
11                  out  $\Delta$ : Set of Values)
12 // Pre:  $c \in C$ ,  $Vars(c) = \{x, y\}$ ,  $b \notin D(y, Q, c)$ 
13 // Post:  $\Delta_1 \subseteq \Delta \subseteq \Delta_2$  with  $\Delta_1 = Inc(c, x, D(y, Q, c)) \cap Cons(c, x, \{b\})$ 
14 //      and  $\Delta_2 = Inc(c, x)$ 

```

Specification 1. The enqueue, post, and valRemove Methods for AC5

The central method of the AC5 algorithm is the `valRemove` method, where the set Δ (called the delta-set in the folklore of CP due the use of the letter Δ in the original AC5 description) is the set of values no longer supported because of the removal of value b in $D(y)$. In this specification, b is a value that is no longer in $D(y)$ and `valRemove` computes the values (x, a) no longer supported because of the removal of b from $D(y)$. Note that values in the queue (for variable y) are still considered in the potential supports as their removal has not yet been reflected in this constraint. We also restrict our attention to values that had the value b in their support (i.e., $(x, a) \in Cons(c, x, \{b\})$). However, we leave `valRemove` the possibility of achieving more pruning (Δ_2), which is useful for monotonic constraints [2].

The AC5 algorithm is depicted in Algorithm 1. Function `propagateQueueAC5` applies `valRemove` on each element of the queue until the queue is empty. Function `initAC5` initializes the queue. Function `post(c, Δ)` computes the inconsistent values of the constraint c . If it removes values in some domains, only the already posted constraints are considered by the `enqueue` call. The constraints not yet posted are not concerned by such removals as they will directly use the current domain of the variables upon posting. The `post` call typically initializes some data structures to be used in `valRemove`. With a slight generalization of the specifications of `post` and `valRemove`, the AC5 algorithm also handles non-binary constraints. AC5 is generic because the implementation of `post` and `valRemove` is left open. Different constraints may have their own implementation of these functions. This allows AC5 to combine, in a single framework, different algorithms such as AC4, AC6, AC7, and AC2001 and to exploit the semantics of the constraints for achieving a better efficiency.

Proposition 1. *Assuming a correct implementation of `post` and `valRemove`, AC5 is correct wrt its specification.*

In AC5, an element (c, x, a) can be put in the queue only once. The size of the queue is thus $O(e.r.d)$ for non-binary CSPs and $O(e.d)$ for binary CSPs. The number of executions of `valRemove` is also bounded by $O(e.r.d)$.

```

1  AC5(in X, C, inout D(X)) {
2  // Pre: (X, D(X), C) is a CSP
3  // Post: D(X) ⊆ D(X)0, (X, D(X), C) equivalent to (X, D(X)0, C)
4  //      (X, D(X), C) is domain consistent
5      initAC5(Q);
6      propagateQueueAC5(Q);
7  }

8  initAC5(out Q) {
9      Q = ∅;
10     C1 = ∅;
11     forall(c in C) {
12         C1 += c;
13         post(c, Δ);
14         forall((x, a) in Δ) {
15             D(x) -= a;
16             enqueue(x, a, C1, Q);
17         }
18     }
19 }

20 propagateQueueAC5(in Q) {
21     while Q != ∅ {
22         select( (c, y, b) in Q) {
23             Q = Q - (c, y, b);
24             valRemove(c, y, b, Δ);
25             forall((x, a) in Δ) {
26                 D(x) -= a;
27                 enqueue(x, a, C, Q);
28             }
29         }
30     }
31 }

```

Algorithm 1. The AC5 Algorithm

Proposition 2. For binary CSPs, if the time complexity of `post` is $O(d^2)$, and the time complexity of `valRemove` is $O(d)$, then the time complexity of AC5 is the optimal $O(e \cdot d^2)$. If the time complexity of `post` is $O(d)$ and the amortized time complexity of all the executions of `valRemove` for each constraint is $O(d)$ (e.g., time complexity of `valRemove` is $O(\Delta)$), then the time and space complexity of AC5 is $O(e \cdot d)$.

We now present AC4 as an instantiation of AC5 by giving the implementation of `post` and `valRemove` (Algorithm 2). `valRemoveAC4` uses a data structure S to record the supports of each value in the different constraints. It is initialized in `postAC4` and satisfies the following invariant at line 21 of Algorithm 1 (AC5).

Let $c \in C$ with $Vars(c) = \{x, y\}$:

- (1.x) $\forall a \in D(x, Q, c) : S[x, a, c] = \{b \in D(y, Q, c) | c(x/a, y/b)\}$
- (2.x) $\forall a \in D(x) : S[x, a, c] \neq \emptyset$

And similarly for y . This invariant ensures the correctness of `valRemoveAC4`. After calling `postAC4`, we also have $\sum_{a \in D(x)} \#S[x, a, c] = \sum_{b \in D(y)} \#S[y, b, c]$ which is $O(d^2)$. The size of the data structure is $O(e \cdot d^2)$.

3 NAC4

NAC4 (Negative AC4), another instance of AC5, is based on forbidden values that are dynamically maintained during the propagation. By NAC4, we mean the AC5 algorithm with the `postNAC4` and `valRemoveNAC4` methods depicted in Algorithms 3 and 4.


```

1  postAC4(in c: Constraint;out Δ: Set of Values) {
2  // Pre: c ∈ C with Vars(c) = {x, y}
3  // Post: Δ = Inc(c, x) ∪ Inc(c, y) + initialization of the S data structure
4  post_varAC4(c, x, Δ1);
5  post_varAC4(c, y, Δ2);
6  Δ = Δ1 ∪ Δ2;
7  }
8  post_varAC4(in c: Constraint;in x: Variable;out Δ: Set of Values) {
9  Δ = ∅;
10 forall(a in D(x)) {
11   S[x, a, c] = ∅;
12   forall(b in D(y) : c(x/a, y/b))
13     S[x, a, c] += b ;
14   if (S[x, a, c]==∅)
15     Δ += (x, a) ;
16 }
17 }
18 valRemoveAC4(in c: Constraint;in y: Variable;in b: Value;
19              out Δ: Set of Values) {
20 // Pre: c ∈ C, Vars(c) = {x, y}, b ∉ D(y, Q, c)
21 // Post: Δ = Inc(c, x, D(y, Q, c)) ∩ Cons(c, x, {b})
22 Δ = ∅;
23 forall(a in S[y, b, c]) {
24   S[x, a, c] -= b ;
25   if (S[x, a, c]==∅ & a in D(x))
26     Δ += (x, a) ;
27 }
28 }

```

Algorithm 2. The post and valRemove Methods for AC4

NAC4 uses a data structure F to record the forbidden values for each value in the different constraints. For a constraint c over x, y , the basic idea is that the value a should be removed from $D(x)$ as soon as the set of forbidden values for (x, a) and the set $D(y)$ are the same. This check can be performed efficiently by (1) reasoning about the sizes of the set of forbidden values for (x, a) and the set $D(y)$, (2) using a data structure sorting the conflict sets by size, and (3) recording the size of the local view of the domains. These data structures are initialized in `postNAC4` and updated in `valRemoveNAC4`. The data structure $F[x, a, c]$ denotes the set of forbidden values for (x, a) and c , $setOfSize[x, k, c]$ denotes the set of values b such that $\#F[x, b, c] = k$, and $localSize[x, c]$ denotes the size of the local view of domain $D(x)$. `valRemoveNAC4` first updates the size of the local view of $D(y)$ and removes b from the $setOfSize$ data structure for variable y (lines 5–7). It then updates the set of forbidden values and $setOfSize$ for each pair $(x, a) \in F[y, b, c]$ (lines 8–13). Finally, it removes the values which are no longer supported, i.e., those values in $setOfSize[x, s, c] \cap D(x)$, where s is the local size of $D(y)$ (lines 14–18).

The data structures satisfy the following invariant at line 21 of Algorithm 2 (AC5). Let $c \in C$ with $Vars(c) = \{x, y\}$:

$$(3.x) \quad \forall a \in D(x, Q, c) : F[x, a, c] = \{b \in D(y, Q, c) \mid \neg c(x/a, y/b)\}$$

$$(4.x) \quad \forall a \in D(x) : F[x, a, c] \subset D(y, Q, c)$$

$$(5.x) \quad \text{setOfSize}[x, k, c] = \{a \in D(x, Q, c) \mid \#F[x, a, c] = k\} (0 \leq k \leq \#D(y, Q, c))$$

$$(6.x) \quad \text{localSize}[x, c] = \#D(x, Q, c)$$

and similarly for y . From these invariants, we have that $F[x, a, c] \subseteq D(y, Q, c)$ at line 24 and the value a must be removed from $D(x)$ if $F[x, a, c] = D(y, Q, c)$. Hence, if $s = \text{localSize}[y, c]$, the algorithm must remove the values in $\text{setOfSize}[x, s, c]$ from $D(x)$. These invariants ensure the correctness of `valRemoveNAC4`.

The size of the data structure is $O(e.d^2)$. In the pruning of the `postNAC4` method, the local view of the size of domain $D(y)$ is $\#D(y)$ since the queue does not contain element of the form (x, \cdot, c) . The complexity of `postNAC4` is $O(d^2)$ and the complexity of `valRemoveNAC4` is $O(d)$. Hence, by Property [2](#) the overall time complexity of NAC4 is $O(e.d^2)$, which has been shown to be the optimal complexity for achieving domain consistency on binary CSPs.

Example 4. We illustrate NAC4 on the following CSP:

$c_1(x, y) = \{(1, 4), (1, 5), (2, 2), (2, 5), (3, 1), (3, 3), (3, 4)\}$, $c_2 : y \neq 4$, $c_3 : y \neq 5$, $D(x) = \{1, 2, 3\}$, and $D(y) = \{1, 2, 3, 4, 5\}$. The execution of `postNAC4`(c_1, Δ) yields $\Delta = \emptyset$ and fills the data structures as follows:

$$\begin{array}{llll} F[x, 1, c_1] & = \{1, 2, 3\} & F[y, 1, c_1] & = \{1, 2\} \\ F[x, 2, c_1] & = \{1, 3, 4\} & F[y, 2, c_1] & = \{1, 3\} \\ F[x, 3, c_1] & = \{2, 5\} & F[y, 3, c_1] & = \{1, 2\} \\ & & F[y, 4, c_1] & = \{2\} \\ & & F[y, 5, c_1] & = \{3\} \\ \text{setOfSize}[x, 1, c_1] & = \emptyset & \text{setOfSize}[y, 1, c_1] & = \{4, 5\} \\ \text{setOfSize}[x, 2, c_1] & = \{3\} & \text{setOfSize}[y, 2, c_1] & = \{1, 2, 3\} \\ \text{setOfSize}[x, 3, c_1] & = \{1, 2\} & \text{setOfSize}[y, 3, c_1] & = \emptyset \\ \text{setOfSize}[x, 4, c_1] & = \emptyset & & \\ \text{setOfSize}[x, 5, c_1] & = \emptyset & & \\ \text{localSize}[x, c_1] & = 3 & \text{localSize}[y, c_1] & = 5 \end{array}$$

`postNAC4`(c_2, Δ) returns $\Delta = \{(y, 4)\}$ and `postNAC4`(c_3, Δ) yields $\Delta = \{(y, 5)\}$, giving $Q = \{(c_1, y, 4), (c_1, y, 5)\}$, $D(x) = \{1, 2, 3\}$, and $D(y) = \{1, 2, 3\}$. Method `valRemoveNAC4`($c_1, y, 4, \Delta$) updates the following variables:

$$\begin{array}{ll} F[x, 2, c_1] & = \{1, 3\} \\ \text{setOfSize}[x, 3, c_1] & = \{1\} \\ \text{setOfSize}[x, 2, c_1] & = \{2, 3\} \\ \text{localSize}[y, c_1] & = 4 \\ \text{setOfSize}[y, 1, c_1] & = \{5\} \end{array}$$

```

1  postNAC4(in c: Constraint;out Δ: Set of Values) {
2  // Pre: c ∈ C with Vars(c) = {x, y}
3  // Post: Δ = Inc(c, x) ∪ Inc(c, y)
4  //   + initialization of the F, setOfSize and localSize data structures
5  post_varNAC4(c, x, Δ1);
6  post_varNAC4(c, y, Δ2);
7  localSize[x, c] = #D(x);
8  localSize[y, c] = #D(y);
9  Δ = Δ1 ∪ Δ2;
10 }
11 post_varNAC4(in c: Constraint;in x: Variable;out Δ: Set of Values) {
12   Δ = ∅;
13   forall(k in 0..#D(y))
14     setOfSize[x, k, c] = ∅;
15   forall(a in D(x)) {
16     F[x, a, c] = ∅;
17     forall(b in D(y) : -c(x/a, y/b))
18       F[x, a, c] += b ;
19     k = #F[x, a, c];
20     setOfSize[x, k, c] += a;
21     if (k==#D(y))
22       Δ += (x, a) ;
23   }
24 }

```

Algorithm 3. The post Algorithm for NAC4

Since $setOfSize[x, 4, c_1] = \emptyset$, $\Delta = \emptyset$, $Q = \{(c_1, y, 5)\}$, $D(x) = \{1, 2, 3\}$, and $D(y) = \{1, 2, 3\}$. $valRemoveNAC4(c_1, y, 5, \Delta)$ updates the following variables:

$$\begin{aligned}
 F[x, 3, c_1] &= \{2\} \\
 setOfSize[x, 2, c_1] &= \{2\} \\
 setOfSize[x, 1, c_1] &= \{3\} \\
 localSize[y, c_1] &= 3 \\
 setOfSize[y, 1, c_1] &= \emptyset
 \end{aligned}$$

Since $setOfSize[x, 3, c_1] = \{1\}$, $\Delta = \{(x, 1)\}$, $Q = \{(c_1, x, 1)\}$, $D(x) = \{2, 3\}$, and $D(y) = \{1, 2, 3\}$. $valRemoveNAC4(c_1, x, 1, \Delta)$ updates the following variables:

$$\begin{aligned}
 F[y, 1, c_1] &= \{2\} \\
 F[y, 2, c_1] &= \{3\} \\
 F[y, 3, c_1] &= \{2\} \\
 setOfSize[y, 2, c_1] &= \emptyset \\
 setOfSize[y, 1, c_1] &= \{1, 2, 3\} \\
 localSize[x, c_1] &= 2 \\
 setOfSize[x, 3, c_1] &= \emptyset
 \end{aligned}$$

The domains are finally $D(x) = \{2, 3\}$ and $D(y) = \{1, 2, 3\}$.

Proposition 3. Let $c \in C$ over $\{x, y\}$. Invariants (3-6.x-y) hold at line 21 of AC5.

Proposition 4. NAC4 is correct and its time and space complexity is $O(e.d^2)$.

```

1  valRemoveNAC4(in c: Constraint;in y: Variable;in b: Value,
2      out  $\Delta$ : Set of Values) {
3      // Pre:  $c \in C$ ,  $Vars(c) = \{x, y\}$ ,  $b \notin D(y, Q, c)$ 
4      // Post:  $\Delta = Inc(c, x, D(y, Q, c)) \cap Cons(c, x, \{b\})$ 
5      localSize[y, c] -- ;
6      k = #F[y, b, c];
7      setOfSize[y, k, c] -= b;
8      forall(a in F[y, b, c]) {
9          F[x, a, c] -= b ;
10         k = #F[x, a, c];
11         setOfSize[x, k + 1, c] -= a;
12         setOfSize[x, k, c] += a;
13     }
14      $\Delta = \emptyset$ ;
15     s = localSize[y, c];
16     forall(a in setOfSize[x, s, c] : a in D(x))
17          $\Delta$  += (x, a);
18 }

```

Algorithm 4. The valRemove method for NAC4

4 Applications

We now review a variety of applications of the principles of maintaining forbidden values.

Sparse AC Constraints. As two instances of AC5, AC4 and NAC4 can be combined, each constraint implementing its AC4 or NAC4 version of the post and valRemove methods. This will be denoted AC5(AC4,NAC4). A nice property of AC5(AC4,NAC4) is that the amortized complexity of all the executions of valRemoveAC4 or valRemoveNAC4 for a constraint is bounded by the number of elements in the data structure S or F in this constraint. We then obtain the following specialization of Proposition 2

Proposition 5. *If a specialization of postAC4 or postNAC4 exploiting the constraint semantics runs in time $O(K)$ for each constraint of a binary CSP, then the time and space complexity of AC5(AC4,NAC4) is $O(e.K)$.*

As a particular case, if S or F can be filled in $O(d)$, a domain-consistency algorithm runs in time $O(e.d)$, as formalized by the following class of constraints.

Definition 5. *A constraint c with $Vars(c) = \{x, y\}$ is positively sparse wrt a domain D iff $\#\{(a, b) \in D^2 \mid c(x/a, y/b)\}$ is $O(\#D)$. The constraint c is negatively sparse wrt D iff $\neg c$ is positively sparse wrt D .*

Example 5. Examples of positively and negatively sparse constraints are bijective constraints ($x + y = k$, where k is a constant), anti-bijective constraints ($x + y \neq k$),

functional constraints ($x = |y - k|$ or $x = y \bmod k$), anti-functional constraints ($x \neq |y - k|$ or $x \neq y \bmod k$), but also include non (anti-)functional constraints such as $|x - y| = k$ and $|x - y| \neq k$. One can also consider congruence constraints, such as $(x + y) \bmod k = 0$ and $(x + y) \bmod k \neq 0$ which are sparse when k is $O(d)$.

Thanks to the genericity of AC5, we can exploit the semantics of the constraints in a specific `postAC4` method for positively sparse constraints and a specific `postNAC4` method for negatively sparse constraints to fill the data structure S or F in $O(d)$ and obtain a time complexity of $O(d)$.

Proposition 6. *For positively and negatively sparse constraints, AC5(AC4,NAC4) can run within a space and time complexity of $O(e.d)$.*

Combining Constraints on the same Variables. Consider now a constraint c over $\{x, y\}$ defined as a boolean combination of constraints $\{c_1, \dots, c_k\}$ on the same variables and assume for simplicity that the number of logical connectors is bounded by k . The constraint c can be posted in AC5(AC4,NAC4) with a complexity of $O(k.d^2)$. The propagation step on this constraint to achieve domain consistency will then run in time $O(K)$, where K is the number of supports.

Example 6. Consider the constraint $c \equiv (c_1 \wedge c_2) \vee (c_3 \wedge c_4)$ where $c_1 \equiv x \neq |y - 2|$, $c_2 \equiv y - 1 \neq x \bmod 2$, $c_3 \equiv x = |y - 1|$, $c_4 \equiv |x - 2| = y$, with $D(x) = \{0, 1\}$ and $D(y) = \{1, 2\}$. Each constraint c_i is domain-consistent, but neither $c_1 \wedge c_2$ nor $c_3 \wedge c_4$ are. Applying AC4 (or NAC4) on c will detect an inconsistency.

In some cases, such as in the above example, it is possible to achieve a better complexity by exploiting both supports and forbidden values. The key idea is that each constraint c_i should use either supports or forbidden values depending on its semantics. Then the individual constraints are combined through logical operators which use the supports and forbidden values to compute their own supports or forbidden values recursively. Table 1 depicts the rules to combine constraints and to compute a data structure S or F for the variables x and y according to the data structure maintained in the subexpressions. The rules are given for variable x but are similar for y . A constraint c_i using an S (resp. F) data structure will be denoted c_i^+ (resp. c_i^-). If the post method applies these rules on c , then the resulting algorithm achieves domain consistency. There is no time or space overhead as all the operations in Table 1 can be performed in time $s_1 + s_2$, where s_i is the size of the data structure (S or F) for c_i . As a particular case, if the time complexity to post each c_i is $O(d)$ (e.g. functional or anti-functional constraint), the time and space complexity of the post constraint for c is $O(k.d)$.

Proposition 7. *Given a set of binary constraints C and a binary constraint c expressed as logic combination of constraints c_1, \dots, c_k with $\text{Vars}(c) = \text{Vars}(c_i)$ ($1 \leq i \leq k$), if the post method for c applies the rules of Table 1 then the resulting AC5(AC4,NAC4) algorithm on $C \cup \{c\}$ achieves domain consistency. If the time complexity of the post methods of the constraints in $C \cup \{c_1, \dots, c_k\}$ is $O(d)$, then the time and space complexity of AC5(AC4,NAC4) applied on $C \cup \{c\}$ is $O((e + k).d)$, with $e = \#C$.*

Table 1. Rules for Combining $c_1(x, y)$ and $c_2(x, y)$

$c^- \equiv \neg c_1^+$	$F[x, a, c] = S[x, a, c_1]$
$c^+ \equiv \neg c_1^-$	$S[x, a, c] = F[x, a, c_1]$
$c^+ \equiv c_1^+ \wedge c_2^+$	$S[x, a, c] = S[x, a, c_1] \cap S[x, a, c_2]$
$c^- \equiv c_1^- \wedge c_2^-$	$F[x, a, c] = F[x, a, c_1] \cup F[x, a, c_2]$
$c^+ \equiv c_1^+ \wedge c_2^-$	$S[x, a, c] = S[x, a, c_1] \setminus F[x, a, c_2]$
$c^+ \equiv c_1^+ \vee c_2^+$	$S[x, a, c] = S[x, a, c_1] \cup S[x, a, c_2]$
$c^- \equiv c_1^- \vee c_2^-$	$F[x, a, c] = F[x, a, c_1] \cap F[x, a, c_2]$
$c^- \equiv c_1^+ \vee c_2^-$	$F[x, a, c] = F[x, a, c_2] \setminus S[x, a, c_1]$

```

1  Boolean isValid(in c: Constraint, in x: Variable, in a: Value)
2  // Pre:  $c \in C$ ,  $Vars(c) = \{x, y\}$ ,  $a \in D(x)$ ,  $D(y) \neq \emptyset$ 
3  // Post: return true iff  $(x, a) \in Valid(c, x, D(y, Q, c))$ 
4  Boolean isEntailed(in c: Constraint)
5  // Pre:  $c \in C$  with  $Vars(c) = \{x, y\}$ ,  $D(x) \neq \emptyset$ ,  $D(y) \neq \emptyset$ 
6  // Post: return true iff  $\forall a \in D(x) : (x, a) \in Valid(c, x, D(y, Q, c))$ 

```

Specification 2. The `isValid` and `isEntailed` Methods

Validity and Entailment. AC5(AC4,NAC4) can be extended to support the `isValid` and `isEntailed` methods (Specification 2). In AC4, a value (x, a) is detected to be valid in c if the size of $S[x, a, c]$ is $\#D(y, Q, c)$. AC4 should then maintain the *setOfSize* and *localSize* data structures of NAC4. In NAC4, a value (x, a) is detected to be valid in c if $F[x, a, c]$ is empty. The invariant of the data structures for both AC4 and NAC4 would then be (1-6.x-y). The theoretical complexity of AC5(AC4,NAC4) is unchanged, while the practical complexity is roughly doubled. AC4 and NAC4 would keep the number of valid values for each constraint c . If the valid values for x in c reaches $\#D(x, Q, c)$, then the constraint is known to be entailed (assuming the domains are non empty). We could also easily extend the `post` and `valRemove` methods to `post(c, Δ^- , Δ^+)` and `valRemove(c, y, b, Δ^- , Δ^+)`, where the extra argument Δ^+ returns the set of new valid values, defined as

$$\Delta^+ = Valid(c, x) \cup Valid(c, y)$$

for `post`, and

$$\Delta^+ = Valid(c, x, D(y, Q, c)) \cap Inc(c, y, \{b\})$$

for `valRemove`. These extended domain consistency algorithms are useful for constraint combinators, reification and in an *Ask & Tell* framework.

Combining Constraints on Different Variables. Achieving domain consistency on a combination of (binary) constraints on different variables is an NP-hard problem. An approximation of domain consistency can be achieved by using the framework proposed in [11], where primitive constraints produce not only the inconsistent values but also the valid ones. Our extended AC5(AC4,NAC4) can be used for combining constraints using the proposed algebra.

5 GNAC4: NAC4 for Non-binary Constraints

This section extends NAC4 to non-binary constraints. It is specified by methods `postGNAC4` and `valRemoveGNAC4` specified in Algorithms 5 and 6. A tuple or vector (v_1, \dots, v_n) is denoted by \mathbf{v} and $\mathbf{v}[x_i]$ denotes the value v_i . We denote $D(X)_{x_i=a}$ the set of tuples \mathbf{v} in $D(X)$ with $\mathbf{v}[x_i] = a$. Let $Y = \{x_1, \dots, x_k\} \subseteq X$. The set of tuples in $D(x_1) \times \dots \times D(x_k)$ is denoted $D(Y)$.

Definition 6. Let c be a constraint with $x, y \in \text{Vars}(c)$, and $D(X) \subseteq B(X)$

$$\begin{aligned} \text{Inc}(c, x, B(X)) &= \{(x, a) \mid a \in D(x) \wedge \forall \mathbf{v} \in B(\text{Vars}(c))_{x=a} : \neg c(\mathbf{v})\} \\ \text{Cons}(c, x, y, b) &= \{(x, a) \mid a \in D(x) \wedge \exists \mathbf{v} : \mathbf{v}[x] = a \wedge \mathbf{v}[y] = b \wedge c(\mathbf{v})\}. \end{aligned}$$

We define $\text{Inc}(c, B(X)) = \bigcup_{x \in \text{Vars}(c)} \text{Inc}(c, x, B(X))$ and similarly for $\text{Cons}(c, y, b)$.

The data structure F is generalized and satisfies the following invariant at line 21 of Algorithm 5 (AC5). Let $c \in C$ with $x \in \text{Vars}(c)$:

$$\begin{aligned} (3'.x) \quad & \forall a \in D(x, Q, c) : F[x, a, c] = \{\mathbf{v} \in D(\text{Vars}(c), Q, c)_{x=a} \mid \neg c(\mathbf{v})\} \\ (4'.x) \quad & \forall a \in D(x) : F[x, a, c] \subset D(\text{Vars}(c), Q, c)_{x=a} \\ (5'.x) \quad & \text{setOfSize}[x, k, c] = \{a \in D(x, Q, c) \mid \#F[x, a, c] = k\} \\ (6'.x) \quad & \text{localSize}[x, c] = \#D(x, Q, c) \end{aligned}$$

and similarly for the other variables of c . The time complexity of `postGNAC4` is $O(r.d^r)$. The time complexity of `valRemoveGNAC4` is $O(r.d^{r-1})$. However, an amortized analysis of GAC4 shows that each element in F can only be removed once, hence a global complexity of $O(e.r.d^r)$ for all the executions of `valRemoveGNAC4`. The complexity of GNAC4 is thus the optimal $O(e.r.d^r)$.

```

1  postGNAC4(in c: Constraint; out Δ: Set of Values) {
2  // Pre: c ∈ C
3  // Post: Δ = Inc(c)
4  //   + initialization of the F, setOfSize and localSize data structures
5  forall(x in Vars(c), k in 1..#D(Vars(c) \ {x}))
6  setOfSize[x, k, c] = ∅;
7  forall(x in Vars(c), a in D(x)) F[x, a, c] = ∅;
8  forall(v in D(Vars(c))): ¬c(v)
9  forall(x in Vars(c))
10 F[x, v[x], c] += v;
11 Δ = ∅;
12 forall(x in Vars(c), a in D(x)) {
13 k = #F[x, a, c];
14 setOfSize[x, k, c] += a;
15 if (k == #D(Vars(c) \ {x}))
16 Δ += (x, a);
17 }
18 }
```

Algorithm 5. The `post` Method for GNAC4

```

1  valRemoveGNAC4(in c: Constraint; in y: Variable; in b: Value,
2      out Δ: Set of Values) {
3      // Pre:  $c \in C, y \in \text{Vars}(c), b \notin D(y, Q, c)$ 
4      // Post:  $\Delta = \text{Inc}(c, D(X, Q, c)) \cap \text{Cons}(c, y, b)$ 
5      localSize[y, c] -- ;
6      k = #F[y, b, c];
7      setOfSize[y, k, c] -= b;
8      forall(v in F[y, b, c]) {
9          forall(x in Vars(c) \ {y}) {
10             a = v[x];
11             F[x, a, c] -= v ;
12             k = #F[x, a, c];
13             setOfSize[x, k + 1, c] -= a; setOfSize[x, k, c] += a;
14         }
15     }
16     Δ = ∅;
17     s = ∏x ∈ Vars(c) localSize[x, c];
18     forall(x in Vars(c) \ {y}) {
19         s1 = s / localSize[x, c];
20         forall(a in setOfSize[x, s1, c] : a in D(x))
21             Δ += (x, a);
22     }
23 }

```

Algorithm 6. The valRemove Method for GNAC4

6 Experimental Results

This section illustrates the benefits of jointly exploiting supports and forbidden values. We evaluated AC4, NAC4, and their combination on CSPs involving the positively and negatively sparse constraints $x = y \bmod k$, $x = |y - k|$, $x + y = k$, $|x - y| = k$, $(x + y) \bmod k = 0$ and their negative version, where k is a constant. Three sets of 20 CSPs were generated: a set with only positive constraints (cPos), a set with only negative constraint (cNeg), and a set with positive and negative constraints (cPosNeg). The results are presented in Table 2. The name cNeg_50_200_10 means that each CSP has 50 variables with a domain $\{0..199\}$, and 10% of constraints between all the pairs of distinct variables. The settings were chosen to avoid trivially consistent or trivially inconsistent CSPs. The constraints were randomly chosen using a uniform distribution. The values k are also determined using a uniform distribution. Each CSP has been solved in Comet using four different consistency algorithms: (1) AC4 for each constraint, (2) NAC4 for each constraint, (3) the combination AC5(AC4,NAC4) using AC4 for positive constraints and NAC4 for negative constraints and (4) the combination AC5(AC4*,NAC*) which is similar to AC5(AC4,NAC4) but uses specialized (linear) post methods exploiting the semantics of the constraints. For CSPs with only positive constraints, AC5(AC4,NAC4) reduces to AC4 and, for CSPs with only negative

Table 2. Comparing AC4, NAC4, AC5(AC4,NAC4) and AC5(AC4*/NAC4*)

	% Consist.	AC4	NAC4	AC5(AC4,NAC4)	AC5(AC4*,NAC4*)
cPos_10_200_01	55%	0.466	19.198	-	0.181
cNeg_50_200_10	100%	27.596	2.381	-	2.027
cPosNeg_50_200_05	53%	21.690	76.073	1.700	1.454

constraints, AC5(AC4,NAC4) reduces to NAC4. The average execution time (in seconds) is reported. We also report the percentage of consistent CSPs in each data set. The inconsistency of the CSPs was always detected in the root node of the search tree. For consistent CSPs, the search is terminated after 1000 fail nodes. The experiments were performed on a single core of a machine with an Intel Core Duo at 2.8GHz with 4GB memory.

For positively sparse constraints, AC4 is much more efficient (speedup of 41) than NAC4, while NAC4 is much more efficient than AC4 (speedup of 11.5) on negatively sparse constraints. This shows the interest of NAC4. Using a specialized post constraint leads to a speedup of 2.57 for AC4 and 1.17 for NAC4. For CSPs combining positively and negatively sparse constraints, NAC4 is 3.5 times slower than AC4, which is explained by the more complicated data structures maintained by NAC4. This last set of CSPs shows the interest of a generic algorithm allowing the combination of different algorithms such as AC4 and NAC4. The speedup of AC5(AC4,NAC4) compared to AC4 is 12.7. This speedup increases to 14.9 when using specialized post methods in AC5(AC4*,NAC4*).

7 Conclusion

This paper proposed the optimal domain-consistency algorithm NAC4 which is not based on supports but dynamically maintains forbidden values during propagation. The ideas behind NAC4 can be combined within the AC5 algorithm with the techniques used in AC4, AC6, and AC2001 for exploiting the semantics of constraints and obtaining greater efficiency. In particular, forbidden values allow AC5 to achieve domain consistency in time $O(ed)$ for classes of constraints in which the number of supports is $O(d^2)$ but the number of forbidden values is $O(d)$. The paper also shows how forbidden values and supports can be used jointly to achieve domain consistency on logical combinations of constraints and to compute validity and entailment of constraints. Experimental results show that the combination of supports and forbidden values can bring significant computational benefits in arc-consistency algorithms.

Future work includes the comparison of AC5(AC4/NAC4) with other AC algorithms, experimental evaluation on other benchmarks, including non-binary CSP instances and extension of NAC4 to handle negative tables represented in a compact way such as in [12][13][14].

Acknowledgment. Many thanks to Jean-Noël Monette for his help in the experimental setting. This research is partially supported by the Interuniversity Attraction Poles Programme (Belgian State, Belgian Science Policy) and the FRFC project 2.4504.10 of the Belgian FNRS (National Fund for Scientific Research).

References

1. Bessière, C.: Constraint propagation. In: Rossi, F., Beek, P.v., Walsh, T. (eds.) *Handbook of Constraint Programming*. Elsevier Science Inc., New York (2006)
2. Van Hentenryck, P., Deville, Y., Teng, C.M.: A generic arc-consistency algorithm and its specializations. *Artif. Intell.* 57(2-3), 291–321 (1992)
3. Bessière, C., Régin, J.C.: Arc consistency for general constraint networks: Preliminary results. In: *IJCAI*, pp. 398–404 (1997)
4. Lecoutre, C.: *Constraint Networks: Techniques and Algorithms*. ISTE/Wiley (2009)
5. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Support inference for generic filtering. In: Wallace, M. (ed.) *CP 2004*. LNCS, vol. 3258, pp. 721–725. Springer, Heidelberg (2004)
6. Mehta, D., van Dongen, M.R.C.: Reducing checks and revisions in coarse-grained MAC algorithms. In: *IJCAI*, pp. 236–241 (2005)
7. Bessière, C., Régin, J.C.: Local consistency on conjunctions of constraints. In: *Proceedings Workshop on Non Binary Constraints on ECAI 1998*, pp. 53–60 (1998)
8. Lhomme, O.: Arc-consistency filtering algorithms for logical combinations of constraints. In: Régin, J.-C., Rueher, M. (eds.) *CPAIOR 2004*. LNCS, vol. 3011, pp. 209–224. Springer, Heidelberg (2004)
9. Van Hentenryck, P., Deville, Y.: The cardinality operator: A new logical connective for constraint logic programming. In: *ICLP*, pp. 745–759 (1991)
10. Van Hentenryck, P., Saraswat, V.A., Deville, Y.: Design, Implementation, and Evaluation of the Constraint Language cc(FD). In: *Constraint Programming: Basics and Trends*, pp. 293–316. Springer, Heidelberg (1994)
11. Bacchus, F., Walsh, T.: Propagating logical combinations of constraints. In: *IJCAI*, pp. 35–40 (2005)
12. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Data structures for generalised arc consistency for extensional constraints. In: *AAAI 2007*, pp. 191–197 (2007)
13. Katsirelos, G., Walsh, T.: A compression algorithm for large arity extensional constraints. In: Bessière, C. (ed.) *CP 2007*. LNCS, vol. 4741, pp. 379–393. Springer, Heidelberg (2007)
14. Cheng, K.C.K., Yap, R.H.C.: Maintaining generalized arc consistency on ad hoc r-ary constraints. In: Stuckey, P.J. (ed.) *CP 2008*. LNCS, vol. 5202, pp. 509–523. Springer, Heidelberg (2008)

Generating Special-Purpose Stateless Propagators for Arbitrary Constraints

Ian P. Gent, Chris Jefferson, Ian Miguel, and Peter Nightingale

School of Computer Science, University of St Andrews, St Andrews, Scotland, UK
{ipg,caj,ianm,pn}@cs.st-andrews.ac.uk

Abstract. Given an arbitrary constraint c on n variables with domain size d , we show how to generate a custom propagator that establishes GAC in time $O(nd)$ by precomputing the propagation that would be performed on every reachable subdomain of $scope(c)$. Our propagators are *stateless*: they store no state between calls, and so incur no overhead in storing and backtracking state during search. The preprocessing step can take exponential time and the custom propagator is potentially exponential in size. However, for small constraints used repeatedly, in one problem or many, this technique can provide substantial practical gains. Our experimental results show that, compared with optimised implementations of the table constraint, this technique can lead to an order of magnitude speedup, while doing identical search on realistic problems. The technique can also be many times faster than a decomposition into primitive constraints in the Minion solver. Propagation is so fast that, for constraints available in our solver, the generated propagator compares well with a human-optimised propagator for the same constraint.

1 Introduction

Constraint models of structured problems often contain many copies of a constraint, which differ only in their scope. English Peg Solitaire, for example, is naturally modelled with a *move* constraint for each of 76 moves, at each of 31 time steps, giving 2,356 copies of the constraint [14]. Efficient implementation of such a constraint is vital to solving efficiency, but choosing an implementation is often difficult. The solver may provide a hand-optimized propagator, otherwise there are two choices: decompose the constraint, or use a table propagator. Decompositions typically introduce extra variables (and so overhead) and/or reduce propagation. In the worst case table propagators take time exponential in the size of their scope. Even hand-optimized propagators the solver provides may not be optimal if they are designed for a more general class of constraints.

The algorithms we give herein generate GAC propagators for arbitrary constraints that run in time $O(nd)$, in extreme cases an exponential factor faster than any table constraint propagator [4,12,7,6,17,15]. As our experiments show, generated propagators can even outperform hand-optimized propagators when performing the same propagation. Our approach is general but in practice does not scale to large constraints as it precomputes domain deletions for all reachable subdomains. It scales easily to 10 Boolean variables, as a case study shows.

2 Theoretical Background

We summarise relevant definitions. For further discussion of propagators see [2].

Definition 1. A *CSP instance*, P , is a triple $\langle V, D, C \rangle$, where: V is a finite set of **variables**; D is a function from variables to their **domains**, where $\forall v \in V : D(v) \subsetneq \mathbb{Z}$; and C is a set of **constraints**. A **literal** of P is a pair $\langle v, d \rangle$, where $v \in V$ and $d \in D(v)$. An **assignment** to any subset $X \subseteq V$ is a set consisting of exactly one literal for each variable in X . Each **constraint** c is defined over a list of variables, denoted $\text{scope}(c)$. A constraint either forbids or allows each assignment to the variables in its scope. An assignment S to V **satisfies** a constraint c if S contains an assignment allowed by c . A **solution** to P is any assignment to V that satisfies all the constraints of P .

Constraint propagators work with subdomain lists, as defined below.

Definition 2. For a set of variables $X = \{x_1 \dots x_n\}$ with original domains $D(x_1), \dots, D(x_n)$, a *subdomain list* for X is a function from variables to sets which satisfies: $\forall 1 \leq i \leq n : S(x_i) \subseteq D(x_i)$. We abuse notation, in a natural way, to write $R \subseteq S$ for subdomain lists R and S iff $\forall 1 \leq i \leq n : R(x_i) \subseteq S(x_i)$. Given a CSP instance $P = \langle V, D, C \rangle$, a **search state** for P is a subdomain list for V .

Backtracking search operates on search states to solve CSPs. During solving, the search state is changed in two ways: branching and propagation. Propagation removes literals from the current search state without removing solutions. Herein, we consider only propagators that establish Generalized Arc Consistency (GAC):

Definition 3. Given a constraint c , a subdomain list S of $\text{scope}(c)$ is **Generalized Arc Consistent (GAC)** if, for every $d \in S(v)$, the literal $\langle v, d \rangle$ is in some assignment which satisfies c and is contained in S .

Any literal that does not satisfy the test in Definition 3 may be removed.

Definition 4. Given a CSP $P = \langle V, D, C \rangle$, a search state S for P and a constraint $c \in C$, The *GAC propagator* for c returns a new search state S' which:

1. For all variables not in $\text{scope}(c)$: is identical to S .
2. For all variables in $\text{scope}(c)$: omits all (and only) literals in S that are in no solution to c , and is otherwise identical to S .

3 Propagator Generation

GAC propagation is NP-hard for families of constraints defined intensionally. For example, establishing GAC on the constraint $\sum_i x_i = 0$ is NP-hard, as it is equivalent to the subset-sum problem [9] (§35.5). However, given a constraint c on n variables, each with domain size d , it is possible to generate a GAC propagator that runs in time $O(nd)$. The approach is to precompute the deletions

performed by a GAC algorithm for every subdomain list for $scope(c)$. The deletions are stored in an array T mapping subdomain lists to sets of literals. The generated propagator reads the domains (in $O(nd)$ time), looks up the appropriate subdomain list in T and performs the required deletions. T can be indexed as follows: for each literal in the initial domains, represent its presence or absence in the sub-domain list with a bit, and concatenate the bits to form an integer.

T can be generated in $O((2^d - 1)^n \cdot n \cdot d^n)$ time. There are $2^d - 1$ non-empty subdomains of a size d domain, and so $(2^d - 1)^n$ non-empty subdomain lists on n variables. For each, GAC is enforced in $O(n \cdot d^n)$ time and the set of deletions is recorded. As there are at most nd deletions, T is size at most $(2^d - 1)^n \cdot nd$.

This algorithm has obvious disadvantages. This preprocessing step can take substantial time and T requires substantial space. However, for small constraints which are used repeatedly, in either one problem or many problems, we shall show how a refinement of this technique can provide substantial practical gains.

4 Generating Tree Propagators

The above approach uses a large data structure, containing all possible subdomain lists. Also, the generated propagator tests the presence of every value in each domain before propagating. We address both problems by using a tree to represent the generated propagator. The tree represents only the subdomain lists that are *reachable*: no larger subdomain fails or is entailed. This improves the average- but not the worst-case complexity. In this section we describe an algorithm that generates a tree-propagator, given any propagator and entailment checker for the constraint in question. First we define tree-propagator.

Definition 5. A *tree-propagator* is a rooted tree $T = \langle V, L, R, r, Prune, Test \rangle$ with vertices V , where $r \in V$ is the root, L is a function mapping vertices to their left child, and R maps vertices to their right child. Two other functions map vertices to prunings and to a literal: $Prune : V \rightarrow 2^{\{(x_i, a) \mid a \in D_i\}}$, and $Test : V \rightarrow \{(x_i, a) \mid a \in D_i\}$.

An execution of a tree-propagator follows a path in T starting at the root r . At each vertex v , the propagator prunes the values specified by $Prune(v)$, and tests if the literal $Test(v) = (x_i, a)$ is in the current domain. If $a \in D(x_i)$, then the next vertex in the path is the left child $L(a)$, otherwise it is the right child $R(a)$. If the relevant child is not present, then the propagator is finished.

SimpleGenTree (Algorithm [11](#)) is a naive algorithm to create a propagator tree given a constraint c and the initial domains D . The algorithm is recursive and builds the tree in depth-first left-first order. Let D_{cur} be the current domain. As a tree-propagator is executed, it tests values to obtain more information about D_{cur} . At a given tree node, each value from the initial domain D may be in D_{cur} , or out, or unknown (not yet tested). SimpleGenTree constructs a sub-domain list SD for each tree node, representing values that are in D_{cur} or unknown. It also constructs $ValsIn$, representing values that are known to be in D_{cur} .

Algorithm 1. SimpleGenTree($c, SD, ValsIn$)

```

1: Deletions  $\leftarrow$  Propagate( $c, SD$ )
2:  $SD' \leftarrow SD \setminus$  Deletions
3: if all domains in  $SD'$  are empty then
4:   return Treenode(Prune=Deletions, Test=Nil, Left=Nil, Right=Nil)
5:  $ValsIn^* \leftarrow ValsIn \setminus$  Deletions
6:  $ValsIn' \leftarrow ValsIn^* \cup \{(x, a) \mid (x, a) \in SD', |SD'(x)| = 1\}$ 
7: if  $SD' = ValsIn'$  then
8:   return Treenode(Prune=Deletions, Test=Nil, Left=Nil, Right=Nil)
   {Pick a variable and value, and branch}
9:  $(y, l) \leftarrow$  heuristic( $SD' \setminus ValsIn'$ )
10: LeftT  $\leftarrow$  SimpleGenTree( $c, SD', ValsIn' \cup (y, l)$ )
11: RightT  $\leftarrow$  SimpleGenTree( $c, SD' \setminus \{(y, l)\}, ValsIn'$ )
12: return Treenode(Prune=Deletions, Test= $(y, l)$ , Left=LeftT, Right=RightT)

```

SimpleGenTree proceeds in two stages. First, it runs a propagation algorithm on SD to compute the prunings required given current domain knowledge. The prunings are stored in the current tree node, and each pruned value is removed from SD and $ValsIn$ to form SD' and $ValsIn^*$. If a domain is empty in SD' , the algorithm returns. If only one value remains for some variable in SD' , the value is added to $ValsIn^*$ to form $ValsIn'$ (because otherwise the domain is empty).

The second stage is to choose a literal and branch. This literal is unknown, ie in SD' but not $ValsIn'$. SimpleGenTree recurses for both left and right branches. On the left branch, the chosen literal is added to $ValsIn$, because it is known to be present in D_{cur} . On the right, the chosen literal is removed from SD . The main terminating condition for the recursion is when $SD' = ValsIn'$. At this point, we have complete knowledge of the current domains: $SD' = ValsIn' = D_{cur}$. The recursion also terminates when a domain is emptied by propagation.

4.1 Generating Code

Algorithm [2](#) (GenCode) generates a program from a tree-propagator via a depth-first, left-first tree traversal. It is called initially with the root r . GenCode creates the body of the propagator function, the remainder is solver specific. In the case of Minion this code is very short and the same for all generated propagators. As an alternative to generating code, it is possible to execute a tree-propagator by traversing the tree at run time. However, in preliminary experiments we found this approach to be roughly 25% slower.

4.2 Correctness

In order to prove the SimpleGenTree algorithm correct, we assume that the Propagate function called on line [1](#) enforces GAC. We need to be careful about what GAC propagators do. Note that, if a GAC propagator produces a domain wipeout, it should also delete all values of all other variables in the constraint. We assume that the Propagate function does this. We also assume that the target

Algorithm 2. GenCode(Tree-propagator T , Vertex v)

```

1: if  $v = \text{Nil}$  then
2:   WriteToCode("NoOperation;")
3: else
4:   WriteToCode("RemoveValuesFromDomains(" + Prune( $v$ ) + ");")
5:   if  $\text{Test}(v) \neq \text{Nil}$  then
6:      $(x_i, a) \leftarrow \text{Test}(v)$ 
7:     WriteToCode("if IsInDomain(" +  $a$  + "," +  $x_i$  + ") then")
8:     GenCode( $T, L(v)$ )
9:     WriteToCode("else")
10:    GenCode( $T, R(v)$ )
11:    WriteToCode("endif;")

```

constraint solver removes all values of all variables in a constraint if our generated propagator empties any individual domain. In practice, constraint solvers often have some shortcut method, such as a special function *Fail* for these situations, but our proofs are slightly cleaner for assuming domains are emptied. Finally we implicitly match up nodes in the generated trees with corresponding points in the generated code for the propagator. Given these assumptions, we will prove that the code we generate does indeed establish GAC.

Lemma 1. *Assuming that the Propagate function in Line 1 establishes GAC, then: given inputs (c, SD, ValsIn) , if Algorithm 1 returns at line 4 or line 8, the resulting set of prunings achieve GAC for the constraint c on any search state S such that $\text{ValsIn} \subseteq S \subseteq SD$.*

Proof. If Algorithm 1 returns on either line 4 or line 8, the set of propagations returned are those generated on Line 1. These deletions achieve GAC propagation for the search state SD .

If the GAC propagator for c would remove a literal from SD , then that literal is in no assignment which satisfies c and is contained in SD . As S is contained in SD , that literal must also be in no assignment which satisfies c and is contained in S . Therefore any literals in S which are removed by a GAC propagator for SD would also be removed by a GAC propagator for S .

We now show no extra literals would be removed by a GAC propagator for S . This is separated into two cases. The first case is if Algorithm 1 returns on line 4. Then GAC propagation on SD has removed all values from all domains. There are therefore no further values which can be removed, so the result follows trivially. The second case is if Algorithm 1 returns on line 8. Then $SD' = \text{ValsIn}'$ on Line 7. This can be reached in one of two cases:

1. $S \setminus \text{Deletions}$ has at least one empty domain. In this case, the returned tree node correctly leads to S having a domain wipeout.
2. $S \setminus \text{Deletions}$ has no empty domains. In this case, any literals added to ValsIn' on line 6 are also in S , as literals are added when exactly one value exists in the domain of a variable in SD , and so this value must also be in S , else there

would be an empty domain in S . Thus we have $ValsIn' \subseteq (S \setminus \text{Deletions}) \subseteq SD'$. But since $ValsIn' = SD'$, we also have $SD' = S \setminus \text{Deletions}$. Since we know SD' is GAC by the assumed correctness of the Propagate function, so is $S \setminus \text{Deletions}$. \square

Theorem 1. *Assuming that the Propagate function in Line 1 establishes GAC, then: given inputs $(c, SD, ValsIn)$, then the code generator Algorithm 2 applied to the result of Algorithm 1 returns a correct GAC propagator for search states S such that $ValsIn \subseteq S \subseteq SD$.*

Proof. We shall proceed by induction on the size of the tree generated by Algorithm 1. The base is that the tree contains just a single leaf node, and this case is implied by Lemma 1. The rest of the proof is therefore the induction step.

By the same argument used in Lemma 1, the Deletions generated on Line 1 can also be removed from S . If applying these deletions to S leads to a domain wipeout, then (as we have assumed) the constraint solver sets $S = \emptyset$, and the propagator has established GAC, no matter what happens in the rest of the tree.

If no domain wipeout occurs, we can progress to Line 9. Again using the same arguments as in Lemma 1, assuming that the Deletions do not cause a domain wipeout in S , then once we get to line 9, we know that $ValsIn' \subseteq S \setminus \text{Deletions} \subseteq SD'$. Since we passed Line 7, we know that $ValsIn' \neq SD'$, and therefore there is at least one value for the heuristic to choose.

There are now two cases. The heuristic value (y, l) is in S , or not.

If $(y, l) \in S$, then the generated propagator will branch left. The propagator generated after this branch is generated from the tree produced by $SimpleGenTree(c, SD', ValsIn' \cup (y, l))$. Since $(y, l) \in S$, we have $ValsIn' \cup (y, l) \subseteq S \setminus \text{Deletions} \subseteq SD'$. Since the tree on the left is strictly smaller, we can appeal to the induction hypothesis that we have generated a correct GAC propagator for $S \setminus \text{Deletions}$. Since we know that Deletions were correctly deleted from S , we have a correct GAC propagator at this node for S .

If $(y, l) \notin S$, the generated propagator branches right. The propagator on the right is generated from the tree given by $SimpleGenTree(c, SD' \setminus (y, l), ValsIn')$ on $S \setminus \text{Deletions}$. Here we have $ValsIn' \subseteq S \setminus \text{Deletions} \subseteq SD' \setminus (y, l)$. As in the previous case, the requirements of the induction hypothesis are met and we have a correct GAC propagator for S .

Finally we note that the set $SD \setminus ValsIn$ is always reduced by at least one literal on each recursive call to Algorithm 1, and can never grow. Therefore we know the algorithm will eventually terminate. With this theorem proved the main result we want is an immediate corollary. \square

Corollary 1. *Assuming the Propagate function correctly establishes GAC for any constraint c , then the code generator Algorithm 2 applied to the result of Algorithm 1 with inputs (c, \emptyset, D) , where D are the initial domains of the variables in c , generates a correct GAC propagator for all search states.*

Lemma 2. *If r is the time a solver needs to remove a value from a domain, and s the time to check whether or not a value is in the domain of a variable, the code generated by Algorithm 2 runs in time $O(nd \max(r, s))$.*

Proof. The execution of the algorithm is to go through a single branch of an if/then/else tree. The tree cannot be of depth greater than nd since one literal is chosen at each depth and there are at most nd literals in total. Furthermore, on one branch any given literal can either be removed from a domain or checked, but not both. This is because Algorithm 1 never chooses a test from a removed value. Therefore the worst case is nd occurrences of whichever is more expensive out of testing domain membership and removing a value from a domain. \square

In some solvers both r and s are $O(1)$, e.g. where domains are stored only in bitarrays. In such solvers our generated GAC propagator is $O(nd)$.

5 Generating Smaller Trees

Algorithm 3 shows the GenTree algorithm. This is an improvement of SimpleGenTree. We present this without proof of correctness, but a proof would be easy since the effect is only to remove nodes in the tree for which no propagation can occur at any subtree.

The first efficiency measure is that GenTree always returns Nil when no pruning is performed at the current node or any of its children. This means that the generated tree will have pruning at all of its leaf nodes. The second efficiency measure is to use an entailment checker. A constraint is *entailed* with respect to a subdomain list SD if every tuple allowed on SD is allowed by the constraint. When a constraint is entailed there is no possibility of further pruning. We assume we have a function ‘entailed’ to check this. The function $\text{entailed}(c, SD)$ is called at the start of GenTree, and also after domains are updated by pruning (line 9). If the constraint is entailed under SD , then no pruning is possible for SD or any sub-domain of it. The value returned is either Nil (if values were pruned at the current node) or a tree node with no children.

To illustrate the difference between SimpleGenTree and GenTree, consider Figure 1. The constraint is very small ($x \vee y$ on Boolean domains) but even so SimpleGenTree generates 7 more nodes than GenTree. The figure illustrates the effectiveness and limitations of entailment checking. Subtree C contains no prunings, therefore it would be removed by GenTree with or without entailment checking. However, the entailment check is performed at the topmost node in subtree C, and GenTree immediately returns (line 2) without exploring the four nodes beneath. Subtree B is entailed, but the entailment check does not reduce the number of nodes explored by GenTree compared to SimpleGenTree. Subtree A is not entailed, however GAC does no prunings here so GenTree will explore this subtree but not output it.

Heuristic. The choice of literal to branch on is very important, and can make a huge difference in the size of the propagator-tree. To minimize the size of the tree, the aim of a heuristic must be to cause Algorithm 3 to return before branching. There are a number of conditions that cause this: entailment (lines 2 and 15); domain wipe-out (line 8); and complete domain information (line 15).

Algorithm 3. Generate Tree-Propagator: $\text{GenTree}(c, SD, \text{ValsIn})$

```

1: if entailed( $c, SD$ ) then
2:   return Nil
3: Deletions  $\leftarrow$  Propagate( $c, SD$ )
4:  $SD' = SD \setminus \text{Deletions}$ 
5: if all domains in  $SD'$  are empty then
6:   return Treenode(Prune=Deletions, Test=Nil, Left=Nil, Right=Nil)
7:  $\text{ValsIn}^* \leftarrow \text{ValsIn} \setminus \text{Deletions}$ 
8:  $\text{ValsIn}' \leftarrow \text{ValsIn}^* \cup \{(x, a) \mid (x, a) \in SD', |SD'(x)| = 1\}$ 
9: if  $SD' = \text{ValsIn}'$  or entailed( $c, SD$ ) then
10:  if Deletions=Nil then
11:    return Nil
12:  else
13:    return Treenode(Prune=Deletions, Test=Nil, Left=Nil, Right=Nil)
    {Pick a variable and value, and branch}
14:  $(y, l) \leftarrow \text{heuristic}(SD' \setminus \text{ValsIn}')$ 
15: LeftT  $\leftarrow$  GenTree( $c, SD', \text{ValsIn}' \cup (y, l)$ )
16: if  $SD'(y) \setminus \{l\} = \emptyset$  then
17:  RightT  $\leftarrow$  Nil
18: else
19:  RightT  $\leftarrow$  GenTree( $c, SD' \setminus \{(y, l)\}, \text{ValsIn}'$ )
20: if LeftT=Nil And RightT=Nil And Deletions= $\emptyset$  then
21:  return Nil
22: else
23:  return Treenode(Prune=Deletions, Test= $(y, l)$ , Left=LeftT, Right=RightT)

```

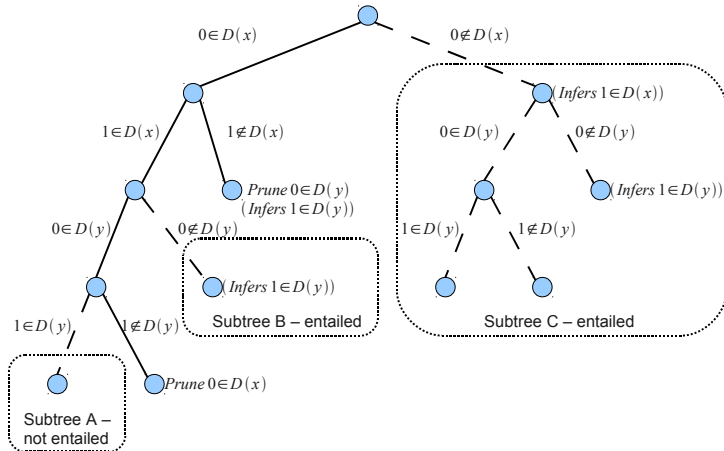


Fig. 1. Example of propagator tree for constraint $x \vee y$ initial domains of $\{0,1\}$. The entire tree is generated by SimpleGenTree (Algorithm 1). The more sophisticated algorithm GenTree (Algorithm 3) does not generate the subtrees A, B and C.

The proposed heuristic greedily attempts to make the constraint entailed. This is done by selecting the literal contained in the greatest number of disallowed tuples of c that are valid with respect to SD .

Implementation. The implementation is recursive and very closely follows the structure of Algorithm 3. It is instantiated with the GAC2001 table propagator 4. The implementation maintains a list of disallowed tuples of c that are valid with respect to SD . This list is used by the entailment checker: when the list becomes empty, the constraint is entailed. It is also used to calculate the heuristic described above. It is implemented in Python and is not highly optimized.

In all the case studies below, we use the solver Minion [11] 0.10. We experiment with 3 generated propagators, in each case comparing against hand-optimized propagators provided in Minion, and also against table constraints where appropriate. All case studies were run with a time out of 24 hours. Instances that took less than 1 hour were run 5 times and the median was taken. In all cases times are given for an 8-core Intel Xeon E5520 at 2.27GHz with 12GB RAM. Minion was compiled with g++ 4.4.1, optimisation level -O3.

Two table constraints were used: Table, which uses a trie data structure with watched literals (as described in [12]), and Lighttable, which uses the same trie data structure but is stateless and uses static triggers. It searches for support for each value of every variable each time it is called.

6 Case Study: English Peg Solitaire

English Peg Solitaire is a one-player game played with pegs on a board. It is Problem 37 at www.csplib.org. The game and a model are described by Jefferson et al [14]. The game has 33 board positions (*fields*), and begins with 32 pegs and one hole. The aim is to reduce the number of pegs to 1. At each step, a peg (A) is jumped over another peg (B) and into a hole, and B is removed. As each move removes one peg, we fix the number of moves in our model to 31.

The model we use is as follows. The board is represented by a Boolean array $b[32, 33]$ where the first index is the time step $0 \dots 31$ and the second index is the field. The moves are represented by Boolean variables $moves[31, 76]$, where the first index is the time step $0 \dots 30$ (where move 0 connects board states 0 and 1), and the second index is the move number, where there are 76 possible moves. The third set of Boolean variables are $equal[31, 33]$, where the first index is the time step $0 \dots 30$ and the second is the field. The following constraints are posted: $equal[a, b] \Leftrightarrow (b[a, b] = b[a + 1, b])$. The board state for the first and last time step are filled in, with one hole at the position we are starting at and one peg at the same position we are finishing at.

For each time step $t \in \{0 \dots 30\}$, exactly one move must be made, therefore constraints are posted to enforce $\sum_i moves[t, i] = 1$. Also for each time step t , the number of pegs on the board is $32 - t$, therefore constraints are posted to enforce $\sum_i b[t, i] = 32 - t$.

The bulk of the constraints model the moves. At each time step $t \in \{0 \dots 30\}$, for each possible move $m \in \{0 \dots 75\}$, the effects of move m are represented by

Table 1. Results on peg solitaire problems

Starting position	Time (s)			Node rate (per s)			Nodes
	Generated	Min	Reified Sumgeq	Generated	Min	Reified Sumgeq	
1	>86389	>86394	>86400	11249	7088	3303	—
2	1.62	2.48	3.10	6338	4140	3312	10,268
4	>86393	>86381	>86369	10986	7514	3926	—
5	879.25	1351.88	3120.26	12964	8431	3652	11,398,210
9	>86400	>86385	>86380	11135	7531	3544	—
10	110.48	167.30	379.22	13456	8886	3920	1,486,641
17	1.49	2.38	3.97	6892	4315	2587	10,269

an arity 7 Boolean constraint. Move m jumps a piece from field f_1 to f_3 over field f_2 . The constraint is as follows.

$$(b[t, f_1] \wedge \neg b[t + 1, f_1] \wedge b[t, f_2] \wedge \neg b[t + 1, f_2] \wedge \neg b[t, f_3] \wedge b[t + 1, f_3]) \Leftrightarrow moves[t, m]$$

Also, a frame constraint is posted to ensure that all fields other than f_1 , f_2 and f_3 remain the same. The constraint states (for all relevant fields f_4) that $equal[t, f_4] = 1$ when $moves[t, m] = 1$.

The arity 7 move constraint is implemented in three ways. The Reified Sumgeq implementation uses a sum to represent the conjunction. The negation of some b variables is achieved with mappers, therefore no auxiliary variables are introduced. The sum constraint is reified to the $moves[t, m]$ variable, as follows: $[(\sum b[t, f_1], \dots, b[t + 1, f_3]) \geq 6] \Leftrightarrow moves[t, m]$.

The Min implementation uses a single `min` constraint, as follows. Again mappers are used for negation. $min(b[t, f_1], \dots, b[t + 1, f_3]) = moves[t, m]$

The Generated propagator was generated by GenTree in 0.14s. The tree has 316 nodes, and the algorithm explored 521 nodes. The propagator was compiled and Minion linked in 16.5s. (For all case studies, we give the time to compile the generated propagator only, plus the time to link Minion, excluding compilation of the rest of Minion.)

Table 1 shows our results for peg solitaire. In all cases the generated propagator outperforms Min by a substantial margin (54% on instance 5), which is perhaps remarkable given that Min is a hand-optimized propagator. For the harder instances, Generated more than repays the overhead of compiling the specialized constraint. The generated propagator outperforms Reified Sumgeq by an even wider margin.

7 Case Study: Low Autocorrelation Binary Sequences

The Low Autocorrelation Binary Sequence (LABS) problem is described by Gent and Smith [13]. The problem is to find a sequence s of length n of symbols $\{-1, 1\}$. For each $k \in \{1 \dots n - 1\}$, the correlation C_k is the sum of the products

Table 2. Results on LABS problems of size 25-30. All times are a median of 5 runs.

n	Time (s)				Search nodes		Nodes per second	
	Generated	Product	Lighttable	Table	Generated, (Light)Table	Product	Generated	Product
25	8.86	11.92	20.54	20.71	206,010	365,470	23252	30660
26	18.20	24.86	51.21	43.03	404,879	731,886	22246	29440
27	41.37	53.27	91.58	90.58	790,497	1,383,351	19108	25969
28	80.67	110.66	182.55	184.72	1,574,100	2,755,212	19513	24898
29	131.91	184.64	326.88	360.36	2,553,956	4,550,121	19361	24643
30	258.58	325.63	711.18	697.31	4,120,335	7,345,259	15934	22557

$s[i] \times s[i+k]$ for all $i \in \{0 \dots n-k-1\}$. The overall correlation is the sum of the squares of all C_k : $\sum_{k=1}^{n-1} (C_k)^2$. This quantity must be minimized.

The sequence is modelled directly, using variables $s[n] \in \{-1, 1\}$. For each $k \in \{1 \dots n-1\}$, and each $i \in \{0 \dots n-k-1\}$, we have a variable $p_k^i \in \{-1, 1\}$ and the **product** constraint $p_k^i = s[i] \times s[i+k]$. For each $k \in \{1 \dots n-1\}$ we have a variable $C_k \in \{-n \dots n\}$. C_k is constrained to be the sum of p_k^i for all i . There are also variables $C_k^2 \in \{0 \dots n^2\}$, and a binary **lighttable** constraint is used to link C_k and C_k^2 . Finally we have $\text{minvar} = \sum_{k=1}^{n-1} C_k^2$, and minvar is minimized. Gent and Smith identified 7 symmetric images of the sequence [13]. We use these to post 7 symmetry-breaking constraints on s . Gent and Smith also proposed a variable and value ordering that we use here.

There are more ternary product constraints than any other constraint in LABS. C_k is a sum of products: $C_k = (s[0] \times s[k]) + (s[1] \times s[k+1]) + \dots$. To test constraint generation on this problem, we combine pairs of product constraints into a single 5-ary constraint: $(s[i] \times s[k]) + (s[i+1] \times s[k+i+1]) = p_k^i$. This allows almost half of the p_k^i variables to be removed.

We compare four models of LABS: *Product*, the model with ternary product constraints; *Generated*, where the new 5-ary constraint has a generated propagator; *Table* and *Lighttable* where the 5-ary constraint is implemented with a table propagator. The Product model does not enforce GAC on the 5-ary constraint. The Generated propagator was generated by GenTree in 0.007s. The algorithm explored 621 nodes and the resulting propagator has 396 nodes. It was compiled and Minion linked in 15.69s.

Table 2 shows our results for LABS sizes 25 to 30. The instances were solved to optimality. The Generated, Table and Lighttable models search the same number of nodes as each other, and exhibit stronger propagation than Product, but their node rate is lower than Product in all cases. The table models are substantially slower than Product. However, Generated is faster than Product, and for the larger instances it more than repays the overhead of compiling the specialized constraint. This is perhaps remarkable when comparing against hand-optimized product and sum constraints.

8 Case Study: Maximum Density Oscillating Life

Conway’s Game of Life was invented by John Horton Conway. The game is played on a square grid. Each cell in the grid is in one of two states (*alive* or *dead*). The state of the board evolves over time: for each cell, its new state is determined by its previous state and the previous state of its eight neighbours (including diagonal neighbours). *Oscillators* are patterns that return to their original state after a number of steps (referred to as the *period*). A period 1 oscillator is named a *still life*.

Various problems in Life have been modelled in constraints. Bosch and Trick considered period 2 oscillators and still lifes [5]. Smith [18] and Chu et al [8] considered the maximum-density still life problem. Here we consider the problem of finding oscillators of various periods. We use simple models for the purpose of evaluating the propagator generation technique rather than competing with the sophisticated still-life models in the literature. However, to our knowledge we present the first model of oscillators of period greater than 2.

The problem of size $n \times n$ (*i.e.* live cells are contained within an $n \times n$ bounding box at each time step) and period p is represented by a 3-dimensional array of Boolean variables $b[n+4, n+4, p]$ indexed (from 0) by position i, j and time step t . To enforce the bounding box, for each t , the rows 0, 1, $n+2$ and $n+3$ are set to 0. Similarly, columns 0, 1, $n+2$ and $n+3$ are set to 0. For a cell $b[i, j, t]$ at time step t , its liveness is determined as follows. The 8 adjacent cells at the previous step are summed: $s = \sum \text{adjacent}(b[i, j, t-1])$, and $(s > 3 \vee s < 2) \Rightarrow b[i, j, t] = 0$, $(s = 3) \Rightarrow b[i, j, t] = 1$, and $(s = 2) \Rightarrow b[i, j, t] = b[i, j, t-1]$. If t is the first time step, then $p-1$ is the previous step, to complete the loop.

We refer to the grid at a particular time step as a *layer*. For each pair of layers, a `watchvecneg` constraint is used to constrain them to be distinct. To break some symmetries, the first layer is lex less than all subsequent layers. Also, the first layer may be reflected horizontally and vertically, and rotated 90 degrees, so it is constrained to be lex less or equal than each of its 7 symmetric images. Finally, all cells in all layers are summed to a variable m which is maximized.

The liveness constraint involves 10 Boolean variables. We generated a propagator using the GenTree algorithm. The algorithm explored 87041 nodes in 45s. The resulting propagator tree has 28351 nodes. The constraint is compiled and Minion linked in 217s, so the total overhead is 262s¹.

The generated propagator is compared to two other implementations. The *Sum* implementation adds an auxiliary variable $s[i, j, t] \in 0 \dots 8$ for each $b[i, j, t]$, and the sum constraint $s[i, j, t] = \sum \text{adjacent}(b[i, j, t-1])$. $s[i, j, t]$, $b[i, j, t-1]$ and $b[i, j, t]$ are linked by a ternary table (`lighttable`) constraint encoding the liveness rules. The *Table* implementation simply encodes the arity-10 constraint as a `table` or `lighttable` constraint.

We used instances with parameters $n \in \{5, 6, 7\}$ and period $p \in \{2, 3, 4, 5, 6\}$. Results are shown in Table 3. In 6 cases, the instances timed out after 24 hours,

¹ In this case the generated constraint was compiled once for Boolean variables only, rather than multiple times for different variable types as is standard in Minion.

Table 3. Time to solve to optimality, for each implementation of the life constraint

n period p		Time (s)				Nodes	Nodes per s, Generated
		Generated	Sum	Lighttable	Table		
5	2	0.04	0.09	0.20	0.22	1,169	29,225
5	3	0.08	0.42	1.34	1.26	5,489	68,613
5	4	0.42	2.38	7.42	6.05	21,906	52,157
5	5	1.09	6.35	21.55	16.66	49,704	45,600
5	6	2.34	11.18	40.00	38.15	71,809	30,688
6	2	0.13	0.67	2.03	2.17	13,631	104,853
6	3	0.93	7.02	19.18	24.59	88,655	95,328
6	4	11.98	75.29	350.19	225.29	886,371	73,988
6	5	124.75	896.97	2779.78	1999.82	6,172,319	49,478
6	6	446.44	3108.18	13929.2	6231.22	16,538,570	37,045
7	2	2.34	13.63	44.57	66.58	316,612	135,304
7	3	18.84	122.13	585.48	377.50	1,905,288	101,130
7	4	366.59	2517.26	12163.6	6706.33	29,194,918	79,639
7	5	9822.84	67014.9	>86393	>86397	564,092,290	50,664
7	6	>86395	>86398	>86398	>86359	—	32,922

but otherwise they were solved to optimality. The three models explored the same number of nodes in all cases.

The generated propagator is substantially faster than the sum implementation. For instance $n = 7$ $p = 5$, Generated is 6.8 times faster than Sum. Also, Sum is faster than Table by a factor of 2 or more. For the four hardest instances that were solved ($n = 6$, $p \in \{5, 6\}$, and $n = 7$, $p \in \{4, 5\}$), the generated propagator more than paid back its 262s overhead. Furthermore, note that the generated propagator is identical in each case: that is the arity 10 constraint is independent of n and p since it depends only on the rules of the game. Therefore the overhead can be amortised over this entire set of runs, as well as any future problems needing this constraint. We can conclude that the generated propagator is the best choice for this set of instances, and by a very wide margin.

9 Related Work

There are a variety of algorithms which achieve GAC propagation for arbitrary constraints, for example GAC2001 [4] and GAC-Schema [3]. The major weakness of these and similar algorithms is that their time complexity for propagation is exponential, with a worst case of (at least) d^n . In GAC2001 and GAC-Schema, constraints presented as allowed tuples have the allowed tuples stored as a simple list. There have been a number of attempts to improve these algorithms by using a more suitable data structure to store the allowed tuples. Many have been used, including tries [12], Binary Decision Diagrams [7], Multi-valued Decision Diagrams [6], skip lists [17] and decision trees [15]. In all cases the worst case complexity is polynomial in the size of the data structure. In some cases the data structure can be much smaller than an explicit list of all allowed tuples,

but the worst case time remains exponential. That is, establishing GAC during search can take time d^n , compared to our worst case of $O(dn)$.

Other improvements to GAC table propagators, such as caching and reusing results [16], have also improved average-case performance, but have not removed the worst-case exponential behaviour.

Constraint Handling Rules is a framework for representing constraints and propagation. Apt and Monfroy [1] have shown how to generate rules to enforce GAC for any constraint, although they state that the rules will have an exponential running time in the worst case. However, such systems can produce very compact sets of propagation rules for some constraints.

The major difference therefore between these techniques and the algorithm in this paper is that our algorithm provides guaranteed polynomial-time execution during search, at the cost of much higher space requirements and preprocessing time than any previous technique. Work in CHR is closest in spirit to our algorithm, but does not guarantee to achieve GAC in polynomial time.

It is possible that techniques from knowledge compilation [10] (in particular prime implicants) could be usefully applied to propagator compilation. However, the rules encoded in a propagator-tree are not prime implicants — the set of known domain deletions is not necessarily minimal. We do not at present know of a data structure which exploits prime implicants and allows $O(nd)$ traversal.

10 Conclusion

We have presented a novel approach to propagating small constraints. The approach is to generate a custom stateless propagator that enforces GAC in $O(nd)$ time. The tradeoff is that the propagator program can be very large — it scales exponentially in the size of the constraint — therefore generating and compiling it is only feasible up to a certain size.

In three case studies, we demonstrated that the propagator generation approach can be highly efficient, compared to table constraints and decompositions. For example, on Life $n = 7$ $p = 4$, the generated constraint is 18 times faster than a table propagator, and 6.9 times faster than a decomposition. Remarkably, generated propagators can even be faster than hand-optimized propagators. For example, 54% faster than a min constraint on peg solitaire 5.

While surprisingly fast, the generated propagators are entirely stateless — there is no state stored between calls, and no local variables. They also do not make use of trigger events, which are often essential to the efficiency of propagators. Therefore we believe there is much scope to improve the scalability of this approach.

Acknowledgements. This research is supported by UK EPSRC Grants no.'s EP/H004092/1 and EP/E030394/1.

References

1. Apt, K.R., Monfroy, E.: Constraint programming viewed as rule-based programming. *Theory and Practice of Logic Programming* 1(6), 713–750 (2001)
2. Bessière, C.: Constraint Propagation. In: *Handbook of Constraint Programming*, pp. 29–83. Elsevier Science Inc., New York (2006)
3. Bessière, C., Régin, J.C.: Arc consistency for general constraint networks: Preliminary results. In: *IJCAI*, vol. (1), pp. 398–404 (1997)
4. Bessière, C., Régin, J.C., Yap, R., Zhang, Y.: An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence* 165, 165–185 (2005)
5. Bosch, R., Trick, M.: Constraint programming and hybrid formulations for three life designs. *Annals of Operations Research* 130, 41–56 (2004)
6. Cheng, K.C., Yap, R.H.: An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints* 15(2), 265–304 (2010)
7. Cheng, K.C.K., Yap, R.H.C.: Maintaining generalized arc consistency on ad-hoc n-ary boolean constraints. In: *Proceeding of the 2006 Conference on ECAI 2006*, pp. 78–82. IOS Press, Amsterdam (2006)
8. Chu, G., Stuckey, P.J., de la Banda, M.G.: Using relaxations in maximum density still life. In: Gent, I.P. (ed.) *CP 2009*. LNCS, vol. 5732, pp. 258–273. Springer, Heidelberg (2009)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 2nd edn. MIT Press/McGraw-Hill (2001)
10. Darwiche, A., Marquis, P.: A knowledge compilation map. *Journal of Artificial Intelligence Research* 17, 229–264 (2002)
11. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast, scalable, constraint solver. In: *Proceedings 17th European Conference on Artificial Intelligence (ECAI 2006)*, pp. 98–102 (2006)
12. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Data structures for generalised arc consistency for extensional constraints. In: *AAAI 2007: Proceedings of the 22nd National Conference on Artificial Intelligence*, pp. 191–197. AAAI Press, Menlo Park (2007)
13. Gent, I.P., Smith, B.M.: Symmetry breaking in constraint programming. In: Horn, W. (ed.) *Proceedings of ECAI-2000*, pp. 599–603. IOS Press, Amsterdam (2000)
14. Jefferson, C., Miguel, A., Miguel, I., Tarim, A.: Modelling and solving english peg solitaire. *Computers and Operations Research* 33(10), 2935–2959 (2006)
15. Katsirelos, G., Walsh, T.: A compression algorithm for large arity extensional constraints. In: Bessière, C. (ed.) *CP 2007*. LNCS, vol. 4741, pp. 379–393. Springer, Heidelberg (2007)
16. Lecoutre, C., Hemery, F.: A study of residual supports in arc consistency. In: *IJCAI 2007: Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 125–130. Morgan Kaufmann Publishers Inc., San Francisco (2007)
17. Lecoutre, C., Szymanek, R.: Generalized arc consistency for positive table constraints. In: Benhamou, F. (ed.) *CP 2006*. LNCS, vol. 4204, pp. 284–298. Springer, Heidelberg (2006)
18. Smith, B.M.: A dual graph translation of a problem in ‘Life’. In: Van Hentenryck, P. (ed.) *CP 2002*. LNCS, vol. 2470, pp. 402–414. Springer, Heidelberg (2002)

Including Ordinary Differential Equations Based Constraints in the Standard CP Framework

Alexandre Goldsztejn², Olivier Mullier^{1,3},
Damien Eveillard¹, and Hiroshi Hosobe³

¹ University of Nantes, CNRS, LINA (UMR 6241), Nantes, France

² CNRS, LINA (UMR 6241), Nantes, France

³ National Institute of Informatics, Tokyo, Japan

Alexandre.Goldsztejn@univ-nantes.fr,

Olivier.Mullier@etu.univ-nantes.fr,

Damien.Eveillard@univ-nantes.fr,

Hosobe@nii.ac.jp

Abstract. Coupling constraints and ordinary differential equations has numerous applications. This paper shows how to introduce constraints involving ordinary differential equations into the numerical constraint satisfaction problem framework in a natural and efficient way. Slightly adapted standard filtering algorithms proposed in the numerical constraint satisfaction problem framework are applied to these constraints leading to a branch and prune algorithm that handles ordinary differential equations based constraints. Preliminary experiments are presented.

1 Introduction

Solving problems involving constraints and ordinary differential equations (ODE) allows numerous applications (e.g. in parameter estimation, control, design). A lot of work has been dedicated to solving such problems ([1,2,3,4,5,6] and references therein). Each of these works proposed a method dedicated to a specific problem, which somehow goes against the paradigm of CSPs that intends separating the problem declaration and the resolution process. However these problems all involve variables on continuous domains and some relations that constrain the variable values thus they should match naturally the numerical CSP framework (NCSP). Such a matching should allow cross-fertilization between the NCSP framework and these specific resolution methods, and expressing and solving more problems involving ODE based constraints. Furthermore, separating the problem declaration and the resolution process has many advantages which have been the basis of the success of the CP framework.

On the other hand, a specialized class of CSPs which includes ODE based constraints was proposed in [7,8], namely Constraint Satisfaction Differential Problems (CSDP). CSDPs include two different kinds of variables: Some functional variables whose values are functions $\mathbf{x}(t)$ from \mathbb{R} to \mathbb{R}^n (corresponding to the trajectories of the ODE) and some real variables, called restriction variables, that correspond to characteristic values of functions $\mathbf{x}(t)$ (for example $\mathbf{x}(0)$

or $\max_{0 \leq t \leq 1} \mathbf{x}(t)$). The constraints of the CSDP include constraints acting on the functional variables (corresponding to the ODE definition) and constraints acting on the restriction variables (like the Value Restriction constraints, e.g. $\|\mathbf{x}(0)\| \leq 1$, and the Maximum Restriction constraints, e.g. $\max_{0 \leq t \leq 1} \mathbf{x}(t) \leq 1$). Thus, although CSDPs are a specific class of CSPs, they do not match the NCSP framework. In particular, they are more complex and less flexible than one could expect (for example restriction variables of CSDPs can only be directly related to the ODE solution through some Restriction constraints, see Subsection 3.2 for an example of problem that cannot be expressed in the CSDP framework).

In the present paper, we show that ODE based constraints can be naturally used in the NCSP framework. This is done by abstracting an ODE by its *solution operator* which is a function that maps an initial condition and a time to the corresponding final state (cf. Section 3). We show that in particular parameter estimation problems and two-point boundary value problems can be expressed homogeneously in this framework (cf. subsections 3.3 and 3.4). The only adaptation of the NCSP framework to solve such constraints involving ODEs is to tune the standard filtering algorithms in order to solve efficiently these constraints. In particular, solving these constraints involve evaluating the solution operator and its derivatives which is very expensive since each evaluation requires simulating the ODE. These solution operator evaluations are performed rigorously for interval inputs using standard methods from interval analysis for integrating ODEs. On the other hand, these methods usually evaluate both the solution operator and its derivatives for the same cost. Therefore, standard NCSPs filtering algorithms have to be tuned in order to evaluate the constraints as rarely as possible, while exploiting all information provided by these evaluations.

Notations. Intervals are denoted using brackets, e.g. $[x]$ is an interval, and the set of closed intervals by \mathbb{IR} . Vectors are denoted by bold symbols, so $[\mathbf{x}]$ is an interval vector (also called a box). Also, $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a vector valued function, which can be viewed as a vector of real valued function, i.e. $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_m(\mathbf{x}))$. Its Jacobian is denoted by $D\mathbf{f}(\mathbf{x}) = (\frac{\partial f_i}{\partial x_j}(\mathbf{x}))_{ij} \in \mathbb{R}^{m \times n}$. The lower and upper bounds of an interval $[x]$ are denoted by $\inf[x] \in \mathbb{R}$ and $\sup[x] \in \mathbb{R}$ respectively. When not confusing, the lower and upper bounds of an interval $[x]$ will be denoted respectively by $\underline{x} \in \mathbb{R}$ and $\bar{x} \in \mathbb{R}$. Inequality for vectors being understood componentwise, $\sup[\mathbf{x}] \leq 0$ means that $[\mathbf{x}]$ contains vectors whose components are all nonpositive. In the case of interval vectors, $\inf[\mathbf{x}] \in \mathbb{R}^n$ and $\sup[\mathbf{x}] \in \mathbb{R}^n$. The width $\text{wid}[x]$ of an interval is $\bar{x} - \underline{x}$, while the width of an interval vector is the maximum of the widths of its components.

2 Interval Analysis for Numerical CSPs

A constraint satisfaction problem (CSP) is a triplet made of a list of variables, a list of domains corresponding to these variables, and a set of constraints. Numerical CSPs are simply characterized by continuous domains, although issues and techniques involved in the resolution of NCSPs and discrete domain CSP are

different. This section presents the basics of NCSPs resolution for constraints that do not involve any ODE.

A solution of a NCSP is often defined as a real vector \mathbf{x} that belongs to the Cartesian product $[\mathbf{x}]$ of the variable domains (i.e. $[\mathbf{x}] = ([x_1], \dots, [x_n])$), whose components correspond to the values of the variables (i.e. $\mathbf{x} = (x_1, \dots, x_n)$) and that satisfies the constraints. The set of its solutions is denoted by $\text{Sol}(\mathcal{P}) \subseteq [\mathbf{x}]$.

2.1 Interval Analysis

The modern interval analysis was born in the 60's with [9] (see [10,11] and extensive references). It allows dealing rigorously with real numbers and functions using machine representable numbers. It therefore plays a key role in the resolution of NCSPs.

The most basic operation of interval analysis is to enclose the image of intervals by a real function. First, arithmetic operations ($+$, $-$, \times and \div) and elementary functions (\exp , \ln , \sin , \cos , etc.) are extended to intervals by $[x] \circ [y] = \{x \circ y : x \in [x], y \in [y]\}$ and $f([x]) = \{f(x) : x \in [x]\}$ respectively. For example, $[\underline{x}, \bar{x}] + [y, \bar{y}] = [\underline{x} + y, \bar{x} + \bar{y}]$ and $\exp([\underline{x}, \bar{x}]) = [\exp(\underline{x}), \exp(\bar{x})]$ (non monotonous functions require more complicated but still simple algorithms). Then, expression compound of these elementary operations can be evaluated to intervals replacing real operations by their interval counterparts. The fundamental theorem of interval analysis states that the interval evaluation of an expression gives rise to an interval enclosure of this function (see e.g. [10]). For example, the interval evaluation of $f(x, y) = xy + \exp(x + y)$ for interval arguments $[x] = [-1, 1]$ and $[y] = [3, 4]$ gives rise to $[3.38, 152.42]$ (rounded to two decimals) which is a superset of $\{f(x, y) : x \in [x], y \in [y]\}$.

The interval evaluation of an expression is the basis for dealing with constraints. For example consider the constraint $f(x, y) = 0$ where f and the domains $[x]$ and $[y]$ are defined as above. The interval evaluation of the expression of f gave rise to an interval that does not contain 0, and therefore proves that the constraint has no solution in the domain hence providing a rigorous filtering process. More elaborated filtering algorithms are presented in Subsection 2.3.

2.2 The Branch and Prune Algorithm

The branch and prune algorithm [12,13] alternates branching and pruning in order to produce two pavings¹ \mathcal{B} and \mathcal{S} , called respectively *boundary boxes* and *solution boxes*. It is described in Algorithm 1 where the pruning is performed by the function $\text{Contract}_{\mathcal{C}}$ whose semantic is

$$\mathbf{x} \in [\mathbf{x}] \wedge (\forall c \in \mathcal{C}, c(\mathbf{x})) \implies \mathbf{x} \in \text{Contract}_{\mathcal{C}}([\mathbf{x}]). \tag{1}$$

So $\text{Contract}_{\mathcal{C}}$ contracts a box in such a way that no NCSP solution is lost, and therefore all the solutions are eventually contained in one box from \mathcal{B} or \mathcal{S} :

$$\text{Sol}(\mathcal{P}) \subseteq (\cup \mathcal{S}) \cup (\cup \mathcal{B}). \tag{2}$$

¹ I.e. sets of boxes which overlap only on their boundaries.

Algorithm 1. Branch and Prune Algorithm.

Input: $C = \{c_1, \dots, c_m\}$, $[\mathbf{x}] \in \mathbb{IR}^n$, $\epsilon > 0$
Output: $S \subseteq \mathbb{IR}^n$, $B \subseteq \mathbb{IR}^n$

```

1  $\mathcal{L} \leftarrow \{[\mathbf{x}]\}$ ;  $B \leftarrow \emptyset$ ;  $S \leftarrow \emptyset$ ;
2 while  $\mathcal{L} \neq \emptyset$  do
3    $([\mathbf{x}], \mathcal{L}) \leftarrow \text{Extract}(\mathcal{L})$ ;
4   if  $\text{IsSolution}_C([\mathbf{x}])$  then  $S \leftarrow S \cup \{[\mathbf{x}]\}$ ;
5   else if  $\text{wid}([\mathbf{x}]) < \epsilon$  then  $B \leftarrow B \cup \{[\mathbf{x}]\}$ ;
6   else
7      $[\mathbf{x}] \leftarrow \text{Contract}_C([\mathbf{x}])$ ;
8     if  $[\mathbf{x}] \neq \emptyset$  then
9        $\{[\mathbf{x}'], [\mathbf{x}'']\} \leftarrow \text{Split}([\mathbf{x}])$ ;
10       $\mathcal{L} \leftarrow \mathcal{L} \cup \{[\mathbf{x}'], [\mathbf{x}'']\}$ ;
11    end
12  end
13 end
14 return  $(S, B)$ ;

```

The solution boxes carry the additional information that the function IsSolution_C , which returns a boolean, evaluates to True on them. The semantic of this function depends on the type of constraints involved in the NCSP. In this paper, we are interested in two kinds of NCSPs that are detailed below.

The first involves only inequality constraints. It will be convenient to group the conjunction of several inequality constraints to one inequality constraint involving a vector valued function: $\mathbf{f}(\mathbf{x}) \leq \mathbf{0}$ with $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$. In this case, the function $\text{IsSolution}_{\{\mathbf{f}(\mathbf{x}) \leq \mathbf{0}\}}([\mathbf{x}])$ returns True if $[\mathbf{f}]([\mathbf{x}]) \leq \mathbf{0}$ and False otherwise. Thus, $\text{IsSolution}_{\{\mathbf{f}(\mathbf{x}) \leq \mathbf{0}\}}([\mathbf{x}])$ is true only if the box $[\mathbf{x}]$ contains only solutions of the constraint. If several such constraints are involved then the conjunction of each constraint solution test can obviously be used. Finally in this case, the branch and prune algorithms outputs pavings that satisfies $\cup S \subseteq \text{Sol}(\mathcal{P})$ in addition to (2). Such pavings are shown in Section 4 (see e.g. Figure 2 page 230).

The second kind of NCSPs involves n equations and n variables (which corresponds to a well constrained system of equations) and possibly additional inequalities. In this case, the NCSP typically has a finite number of solutions and the function $\text{IsSolution}_C([\mathbf{x}])$ returns true only if the existence of one solution in $[\mathbf{x}]$ is proved. This existence proof is carried out using the multidimensional interval Newton operator (4) which is introduced in Subsection 2.3. Therefore uniqueness of this solution in $[\mathbf{x}]$ is also proved.

2.3 Filtering Algorithms for NCSP

Standard filtering algorithms used in the NCSP framework must be slightly adapted in order to be efficiently used for constraints involving ODEs. These constraints are characterized by the fact that evaluating a function involving an ODE is very expensive, but only a small extra work is necessary to evaluate the

Jacobian of this function (see Section 3 for details). Thus the filtering algorithms presented in this section also classically used in the context of NCSP may turn out not to be the most efficient ones when constraints involve no ODEs.

Conjunction of inequality constraints. Two filtering algorithms will be used for such constraints $\mathbf{f}(\mathbf{x}) \leq \mathbf{0}$: The simple interval enclosure test and the unidimensional interval Newton². The interval enclosure test simply consists of computing an interval enclosure of \mathbf{f} and checking whether or not it allows rejecting the whole box: If $\inf[\mathbf{f}](\mathbf{x}) \leq \mathbf{0}$ is not satisfied then the whole box can be rejected. The unidimensional interval Newton allows removing a slice of \mathbf{x} as follows: The domain $[x_j]$ of the variable x_j is contracted to the new domain $[x'_j]$ applying the unidimensional interval Newton for each component of $\mathbf{f}(\mathbf{x}) \leq \mathbf{0}$:

$$[x'_j] = \bigcap_{i \in \{1, \dots, m\}} \left(\tilde{x}_j - \frac{1}{[a_{ij}]} ([b_i] + [0, \infty] + \sum_{k \neq j} [a_{ik}]([x_k] - \tilde{x}_k)) \right) \cap [x_j] \quad (3)$$

where $[A] = [D\mathbf{f}](\mathbf{x})$, $[\mathbf{b}] = [\mathbf{f}](\tilde{\mathbf{x}})$ for some $\tilde{\mathbf{x}} \in \mathbf{x}$, usually the midpoint of \mathbf{x} . The addition of the interval $[0, \infty]$ allows applying the interval Newton, which is originally defined for equality constraints, to inequality constraints: Indeed, $f(x) \leq 0$ is equivalent to $0 \in f(x) + [0, \infty]$ and the interval Newton for equality constraint is actually applied to the latter.

It is worth noting that in order to apply these two filtering algorithms, only interval enclosures of \mathbf{f} over \mathbf{x} and over $\tilde{\mathbf{x}}$, and of $D\mathbf{f}$ over the whole domain are required.

Well constrained systems of equations. The simple interval enclosure test is also applied to constraints $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ by rejecting the whole box if $0 \in [\mathbf{f}](\mathbf{x})$ is not satisfied. We will also use the Krawczyk version of the multidimensional interval Newton (see e.g. [10]) that allows contracting the whole domain \mathbf{x} to the new domain \mathbf{x}' as follows:

$$[\mathbf{x}'] = ((I - C[A])[\mathbf{x}] + C[\mathbf{b}]) \cap [\mathbf{x}] \quad (4)$$

where $[A]$ and $[\mathbf{b}]$ are defined like in (3), and $C = (\text{mid}[A])^{-1}$.

Remark 1. The multidimensional interval Newton operator is used for both filtering non consistent vectors and proving the existence of solutions (see [10] for details). Thus only one application of the multidimensional interval Newton is performed for both purposes, while this optimization is not explicitly shown in Algorithm 1 for clarity.

It is worth noting that again in order to apply the filtering algorithms dedicated to well constrained systems of equations, only interval enclosures of \mathbf{f} over

² Usually, the unidimensional interval Newton is improved encapsulating it into the box-consistency filtering (cf. [14] and references therein). However, the box-consistency filtering requires many function evaluations, which makes it too expensive when ODEs are involved.

$[\mathbf{x}]$ and over $\tilde{\mathbf{x}}$, and of Df over the whole domain are required. Therefore, all we will need to be able to apply these filtering algorithms to ODE based constraints is to be able to compute these interval enclosures in the case of constraints involving ODEs. This is done by abstracting the ODEs by their solution operator, as shown in the next section.

3 Including ODE Based Constraints in CSPs

We consider herein an ODE $\mathbf{x}'(t) = \mathbf{h}(\mathbf{x}(t))$ where $\mathbf{h} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is supposed to be enough differentiable so that interval integrator can solve it (see e.g. [15] for an introduction to ODEs). The *solution operator* of this ODE is introduced in Subsection 3.1, while the remaining subsections show how to use this solution operator to model different problems.

3.1 The ODE Solution Operator and Its Derivatives

The solution operator is introduced noting that the ODE maps an initial condition $\mathbf{x}(t_0) \in \mathbb{R}^n$ and a duration $t \in \mathbb{R}$ to an unique vector³ $\mathbf{x}(t)$. Therefore, the ODE defines an operator $\Phi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n$ characterized by

$$\Phi(\mathbf{x}(t_0), t) = \mathbf{x}(t_0 + t) \quad (5)$$

called the ODE solution operator. It allows to abstract the simulation of the ODE into a simple function evaluation. On the other hand, the evaluation of the solution operator of course requires to integrate the ODE, thus each evaluation of Φ is computationally very expensive.

The Jacobian $D\Phi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^{n \times (n+1)}$ of the solution operator is also very useful as it allows to quantify the sensitivity of Φ . For convenience, it is split into two submatrices $D_{\mathbf{x}}\Phi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^{n \times n}$ and $D_t\Phi : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^{1 \times n}$, the latter being actually equal to $\mathbf{h}(\Phi(\mathbf{x}, t))$. The former is usually computed solving the ODE first variational equation, which is a linear non-autonomous ODE of dimension n^2 . Therefore, evaluating Φ and $D\Phi$ turns out to solving an ODE of dimension $n^2 + n$.

Standard integrators coming from numerical analysis (e.g. Runge-Kutta, Adams methods, etc., see e.g. [15] for details) compute approximations of the ODE solution. Therefore, they can be used to evaluate approximately Φ and $D\Phi$. On the other hand, interval integrators (see [16] for review and references therein for the theory of interval integrators) allow enclosing rigorously the solution of the ODE for interval initial conditions. Therefore interval integrators give rise to interval enclosures of Φ and $D\Phi$. This is the key point of our approach: In order to apply the NCSP framework to ODE based constraints, we need interval enclosures of the solution operator and its Jacobian, these interval enclosures being computed by interval ODE integrators.

³ Existence and uniqueness are assumed here, and follow from some hypothesis on the function ϕ that are usually verified. While the concept of solution operator can be generalized considering a domain different than \mathbb{R}^n , the interval integrators used in the sequel rigorously prove existence and uniqueness.

Remark 2. The work [17] proposes to use constraints techniques to help integrating ODE. However, this kind of techniques cannot be used directly in our framework since it does not provide any enclosure of the solution operator derivatives.

In theory an interval ODE integrator can compute $[\Phi](\mathbf{x}, [t])$, $[\Phi](\text{mid}[\mathbf{x}], [t])$ and $[D\Phi](\mathbf{x}, [t])$ in simulation for approximately the same cost than computing only $[\Phi](\mathbf{x}, [t])$ (see [18,19]). This is perfectly fitted to the application of the different contractors used in NCSP resolution presented in Section 2.3.

In the next subsections, we show how the solution operator allows expressing ODE based constraints modeling different problems. It can be note already that any problem that can be expressed in the CSDP framework can also be expressed using the solution operator (this is trivial since the solution operator is actually equivalent to the ODE itself so any restriction acting on the ODE can be expressed using the solution operator).

3.2 An Academic Design Problem

A simple academic design problem first shows the expressiveness of the NCSP framework with ODE based constraints. We consider for that a 2D ODE $\mathbf{x}'(t) = \mathbf{h}(\mathbf{x}(t))$. We are looking for the points \mathbf{a} and \mathbf{b} belonging to the radius 1 circle centered on $(1, 0)$ such that the state $\mathbf{x}(t_f)$ at a fixed time t_f of the ODE starting at \mathbf{a} (i.e. $\mathbf{x}(0) = \mathbf{a}$), and the points \mathbf{a} and \mathbf{b} form an equilateral triangle. In the case where the ODE is $\mathbf{x}'(t) = A\mathbf{x}(t)$ with

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \tag{6}$$

and thus the solution operator $\Phi(\mathbf{x}, t) = \exp(tA)$ is a rotation of angle t , and $t_f = \frac{\pi}{2}$, the solution can be found graphically as depicted in Figure 1.

This problem is expressed by the following simple NCSP: Variables are two angles θ and τ that define the position of \mathbf{a} and \mathbf{b} on the circle (i.e. $\mathbf{a}_\theta = (\cos(\theta), \sin(\theta))$ and $\mathbf{b}_\tau = (\cos(\tau), \sin(\tau))$) and the domains of these variables are $[-\pi, \pi]$. There are two constraints relating the distances between the three points :

$$\|\mathbf{a}_\theta - \Phi(\mathbf{a}_\theta, t_f)\|^2 - \|\mathbf{a}_\theta - \mathbf{b}_\tau\|^2 = 0 \tag{7}$$

$$\|\mathbf{b}_\tau - \Phi(\mathbf{a}_\theta, t_f)\|^2 - \|\mathbf{a}_\theta - \mathbf{b}_\tau\|^2 = 0 \tag{8}$$

where $\Phi : \mathbb{R}^2 \times \mathbb{R} \longrightarrow \mathbb{R}^2$ is the solution operator of the ODE. The expression of this geometric problem as a NCSP is really clear (even clearer than its original English description). Note that this problem cannot be expressed in the CSDP framework as variables are neither CSDP solution variables nor restriction variables⁴. Using the chain rule the constraints can be automatically differentiated,

⁴ It could be possible to add a third kind of variables in the CSDP framework to handle this NCSP. However, the CSDP framework is already made quite complex because of these two kinds of variables and introducing a third kind of variables would make it even more complex.

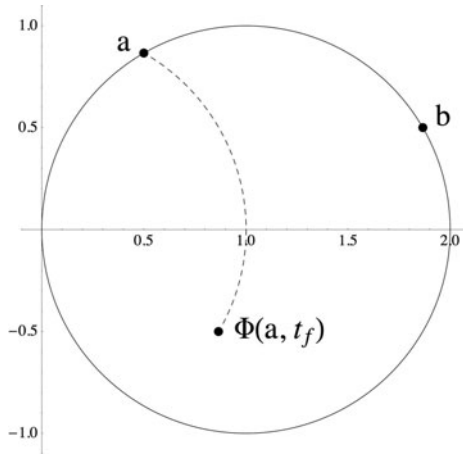


Fig. 1. Graphical solution of the NCSP of Subsection 3.2. The dashed line shows the ODE flow starting at **a** which is abstracted by the ODE solution operator in the NCSP definition.

hence allowing applying the contractors based on the interval Newton operator. Thus this problem can now be solved using the standard methods dedicated to NCSPs.

3.3 Parameter Estimation Problems

The parameter estimation problem represents a special case of finding the initial conditions that satisfy some constraints (usually coming from measurements or observations). Formally, given an ODE $\mathbf{x}'(t) = \mathbf{h}(\mathbf{x}(t))$ and some measurements $(t_i, [\mathbf{m}_i])$ for $i \in \{1, \dots, p\}$, the problem consists of finding the initial values such that the state satisfies $\mathbf{x}(t_i) \in [\mathbf{m}_i]$ for $i \in \{1, \dots, p\}$. Using the solution operator, this problem is easily cast to a standard NCSP (V, D, C) where: $V = \mathbf{x} = (x_1, \dots, x_n)$ are the initial conditions we search, $D = [\mathbf{x}] = ([\mathbf{x}_1], \dots, [\mathbf{x}_n])$ is the initial search region for the initial conditions and

$$C = \{\Phi(\mathbf{x}, t_1) \in [\mathbf{m}_1], \dots, \Phi(\mathbf{x}, t_p) \in [\mathbf{m}_p]\}. \tag{9}$$

Each constraint represents two vectorial inequalities which can be handled using the evaluation test and the unidimensional interval operator as explained in Section 2.

However, an efficient handling of the conjunction of constraints C require an important optimization: When evaluating the solution operator for \mathbf{x} and t_i , the interval integrator performs a simulation from 0 to t_i . Therefore, if the interval evaluation of the solution operator for different times are performed independently then simulations are unnecessarily repeated. Therefore, only one simulation from 0 to t_p must be performed to evaluate the solution operator and its Jacobian for all time measures.

3.4 Two-Point Boundary Value Problems

A two-point boundary value problem (TPBVP) consists of an ODE $\mathbf{x}'(t) = \mathbf{h}(\mathbf{x}(t))$ and n equality constraints g_i that relate $\mathbf{x}(t_0)$ and $\mathbf{x}(t_1)$. Solving the TPBVP consists in finding trajectories $\mathbf{x}(t)$ satisfying the ODE and the constraints $g_i(\mathbf{x}(t_0), \mathbf{x}(t_1)) = 0$. As the trajectory is completely defined by its initial condition, the TPBVP actually consists in finding the initial conditions that give rise to such trajectories. Such a problem perfectly fits the NCSP framework since these initial conditions are the solutions of the NCSP (V, D, C) where: $V = \mathbf{x} = (x_1, \dots, x_n)$ are the initial conditions we search, $D = [\mathbf{x}] = ([\mathbf{x}_1], \dots, [\mathbf{x}_n])$ is the initial search region for the initial conditions and $C = \{c_1(\mathbf{x}) = 0, \dots, c_n(\mathbf{x}) = 0\}$ with

$$c_i(\mathbf{x}) = g_i(\mathbf{x}, \Phi(\mathbf{x}, t_1 - t_0)). \tag{10}$$

Note that once $[\Phi]([\mathbf{x}], t_1 - t_0)$, $[\Phi](\text{mid}[\mathbf{x}], t_1 - t_0)$ and $[D\Phi]([\mathbf{x}], t_1 - t_0)$ are evaluated then $[c_i]([\mathbf{x}])$, $[c_i](\text{mid}[\mathbf{x}])$ and $[Dc_i]([\mathbf{x}])$ directly follow from (10). In particular using the chain rule we obtain

$$Dc_i(\mathbf{x}) = \left. \frac{\partial g(\mathbf{u}, \mathbf{v})}{\partial \mathbf{u}} \right|_{(\mathbf{x}, \mathbf{y})} + \left. \frac{\partial g(\mathbf{u}, \mathbf{v})}{\partial \mathbf{v}} \right|_{(\mathbf{x}, \mathbf{y})} D_{\mathbf{x}}\Phi(\mathbf{x}, t_1 - t_0) \tag{11}$$

with $\mathbf{y} = \Phi(\mathbf{x}, t_1 - t_0)$. The expression (11) can be evaluated for interval arguments using $[\Phi]([\mathbf{x}])$ and $[D\Phi]([\mathbf{x}])$.

Remark 3. In some cases the number of variables of the problem can be reduced, which ease its resolution by a branch and prune algorithm. This is the case typically when one constraint g_i involves only the state at $t = t_0$ and can be solved formally. For example let $g_1(\mathbf{u}, \mathbf{v}) = u_1 - u_2$ and $g_2(\mathbf{u}, \mathbf{v}) = v_1 - v_2$ so the TPBVP to be solved is $x_1(t_0) = x_2(t_0)$ and $x_1(t_1) = x_2(t_1)$. So we need only to perform the search on $x_1(0)$ since $x_2(0)$ is a function of the former. So the NCSP we solve will actually be: $V = \{x\}$, $D = \{[x]\}$ and $C = \{\Phi((x, x), t_1) - \Phi((x, x), t_0) = 0\}$ which is simpler than the original NCSP involving the constraints (10).

4 Experiments

Experiments have been carried out on a 2.53 GHz Intel Dual Core. We used the interval ODE integrator available as part of the CAPD library⁵. In theory $[\Phi]([\mathbf{x}], [t])$, $[D\Phi]([\mathbf{x}], [t])$ and $[\Phi](\text{mid}[\mathbf{x}], [t])$ can be computed during one single integration. However, the CAPD integrator does not offer this possibility so we performed two integrations independently, thus approximately doubling the computational time. Also, the CAPD integrator is optimized for small initial conditions (which is required for the usual usage of this integrator related to chaotic dynamical systems investigation) so we obtained very sharp enclosures of solutions, but poorer performances for exploring large domains.

⁵ CAPD is available at <http://capd.i1.uj.edu.pl/>

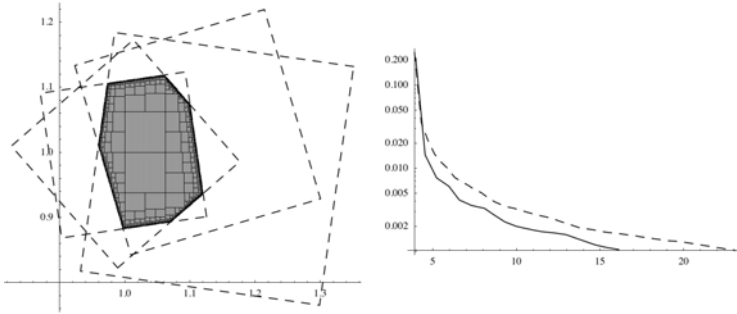


Fig. 2. Left: Paving obtained for the academic problem of Section 4.1 (solution and boundary boxes are in gray and black respectively). Right: Volume of the boundary boxes vs time for the interval enclosure test (dashed) and the Newton contractor (plain).

4.1 Parameter Estimation Problems

Academic Example. It is a simple problem for which we can formally compute the solution set. We look for the initial conditions $\mathbf{x}(0) \in ([0, 2], [0, 2])$ for which the trajectories of the ODE $\mathbf{x}'(t) = A\mathbf{x}(t)$ with

$$A = \begin{pmatrix} 0.1 & 1 \\ -1 & 0.1 \end{pmatrix} \tag{12}$$

(trajectories are spirals that unroll toward infinity) belong to the box $[\mathbf{m}_i] = \mathbf{m}_i \pm 0.25$ at time t_i . In that purpose, we dispose of the following values:

t_i	3	5	7	8
\mathbf{m}_i	$(-1.34, -1.52)$	$(-1.11, 2.24)$	$(2.84, 0.19)$	$(1.87, -2.52)$

As the ODE is linear, its trajectories can be formally computed using the matrix exponential: $\mathbf{x}(t) = \exp(tA)\mathbf{x}(0)$. Therefore, each constraint that forces the trajectory to be in a specific box at time t_i actually implies that the initial condition belongs to the parallelepiped $\{\exp(t_i A)^{-1} \mathbf{u} : \mathbf{u} \in \mathbf{m}_i \pm 0.25\}$. These parallelepipeds are represented in dashed lines on the left hand side diagram of Figure 2 together with the paving computed by our algorithm. The right hand side diagram of Figure 2 shows the total volume of the boundary boxes (vertical axis) that decreases with time (horizontal axis). The plain curve represents the timings obtained using both the interval enclosure test and the contractor based on the interval Newton, while the dashed curve corresponds to the interval enclosure test alone. We can see that the Newton based contractor improves the resolution, although it could be used more efficiently: Indeed, it could also be used to compute inner boxes (cf. [20]) or in conjunction with parallelotope domains (cf. [21]) although this framework needs to be extended to inequality constraints for this purpose.

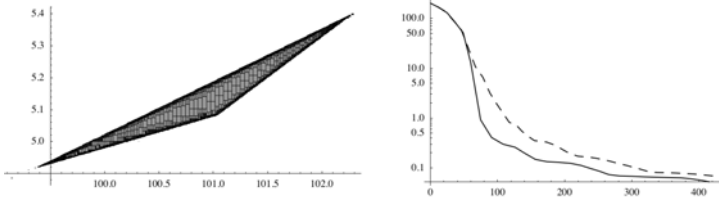


Fig. 3. Left: Paving obtained for the kinetic parameters estimation of an enzymatic reaction of Section 4.1 (solution and boundary boxes are in gray and black respectively). Right: Volume of the boundary boxes versus time for the interval enclosure test (dashed) and the Newton contractor (plain).

Estimating Kinetic Parameters of an Enzymatic Reaction. In [22] was reported the important potential interest of rigorously simulating the following kinetic enzymatic reaction:

$$\begin{aligned} s'(t) &= -\frac{V_{\max} s(t)}{k_s + s(t)} \\ p'(t) &= \frac{V_{\max} s(t)}{k_s + s(t)} \end{aligned} \tag{13}$$

However, the used implementation of HybridCC [23,24] could not integrate this ODE sharply enough to provide interesting numerical evidences. The NCSP framework proposed herein uses CAPD which is able to integrate sharply the ODE (13). Thus, we are in position to perform some parameter estimation of this model. Some measurements are provided for parameter estimation in [25,26], however the duration of the experiment is too long for CAPD to integrate it sharply enough for large initial conditions. Thus, we have prepared the following set of measurements by simulating the system for $s(0) = 25$, $p(0) = 0$, $V_{\max} = 100$ and $k_s = 5$ and added a random noise of amplitude 0.1. We obtained the following measurements:

t_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
p_i	8.01	15.32	21.01	23.92	24.74	24.96	24.97	25.02	24.95	24.91

The parameter estimation problem consists in finding the parameter values (V_{\max}, k_s) in the initial domain $([90, 110], [0, 10])$ which satisfy $p(t_i) \in p_i \pm 0.1$. Results are shown in Figure 3. We see again that the interval Newton based contractor improves the resolution in spite of the possible improvements of its integration with the solution operator evaluation.

4.2 Two-Point BVPs

We have solved two boundary value problems that were studied in [6]. The method proposed in [6] consists of a bisection algorithm that implements a specific filtering algorithm on Taylor models (see [27] for details on the Taylor model method used in [6]). On the one hand, using Taylor models allows integrating

Table 1. Solutions for the catalytic reaction in a flat particle problem computed from the initial interval $[0, 1]$

λ	Solution enclosure	Width	Time (s)
0.05	0.97034556001404[65, 88]	2.2×10^{-15}	1.1
0.1	0.9226804137526[691, 733]	4.2×10^{-15}	3.7
	0.5058725840206[640, 786]	1.5×10^{-14}	
	0.06446821272269[81, 149]	6.7×10^{-15}	
0.15	0.01655884279376[13, 30]	1.6×10^{-15}	5.3

sharply larger initial conditions than the algorithm implemented in CAPD so larger initial domains can be used than in our implementation (cf. the tubular reactor model problem). On the other hand, the NCSP framework allows to compute sharper enclosures and proving rigorously the existence of solutions which is an important issue in BVP resolution. This detailed in the following.

Catalytic Reaction in a Flat Particle. Under some assumptions, the catalytic reaction of a particle can be modeled by the following BVP (cf. [28]):

$$\begin{aligned}
 x_1'(t) &= x_2(t) \\
 x_2'(t) &= \lambda x_1(t) \exp\left(\frac{\gamma\beta(1-x_1(t))}{1+\beta(1-x_1(t))}\right) \\
 x_2(0) &= 0 \\
 x_1(1) &= 1.
 \end{aligned} \tag{14}$$

This is naturally expressed as a NCSP as shown in Section 3.4. As in [6], we solved the problem for $\gamma = 20$, $\beta = 0.4$, three values of λ and the initial search interval $x_1(0) \in [0, 1]$. Table 1 reports the results obtained. We have found the same solutions in approximately the same computation time as in [6]. However, the enclosures obtained by our algorithm are much sharper, while we have proved the existence of one unique solution in each of them. This is an important advantage of the usage of the interval Newton as usually implemented in the NCSP framework.

Tubular Reactor Model. The following simplified model of a tubular reactor was studied in [29,6]:

$$\begin{aligned}
 x_1'(t) &= x_2(t) \\
 x_2'(t) &= 6\left(x_2(t) - 0.05(1 - x_1(t)) \exp\left(\frac{10x_1(t)}{1+0.5x_1(t)}\right)\right) \\
 x_2(0) &= 6x_1(0) \\
 x_2(1) &= 0.
 \end{aligned} \tag{15}$$

In [6] the initial search domain was $x_1(0) \in [0, 1]$. However, we found that a trajectory starting at $x_1(0) = 0$ and $x_2(0) = 0$ probably converges to a singularity in finite time, which prevents any further integration[6]. The bisection

⁶ This behavior was pointed out by Daniel Wilczak in a personal communication.

Table 2. Solutions for the tubular reactor model computed in 30 seconds from the initial interval $[0.01, 0.04]$

Solution enclosure	Width
0.010042462528303[8, 9]	2.8×10^{-17}
0.01844542857635[59, 68]	7.6×10^{-16}
0.03738712388979[57, 179]	1.3×10^{-14}

algorithm proposed in [6] is able to get rid of this convergence to a singularity by enforcing the constraint $x_1(t) \in [0, 1]$ for all t , which is justified by physical considerations. This additional constraint strongly ease the resolution of the BVP, but we have not yet been able to integrate this constraint in our framework. This together with the fact that CAPD is optimized for small initial conditions (while the ODE integrator used in [6] better integrates larger initial conditions) restricted us to a smaller initial domain: Table 2 shows the enclosures computed by our algorithm starting from the initial domain $x_1(0) \in [0.01, 0.04]$ (which contains all the BVP solutions). We have found the same solutions as in [6]. However, five intervals were found in [6] and the authors conjectured that three of them were corresponding to the same solution (because they were very small neighbors). Our algorithm gave the rigorous proof that a unique solution lies in each intervals hence proving that this BVP actually has three solutions, and provided sharper enclosures of these solutions. These are the typical advantages of the NCSP framework.

5 Conclusion

We have shown that abstracting an ODE by its solution operator allows including naturally any ODE based constraint into the standard framework of numerical CSPs (NCSP). This permits the homogenizing and generalizing of several state of the art algorithms, like for example those dedicated to parameter estimation and boundary value problems. The expression of problems involving ODE in this framework presents many advantages: First it allows to separate the problem declaration and its resolution, which is a well known advantage of the CP framework. Second, the resolution of these problems can benefit from present and future efficient methods designed in the NCSP framework.

Our current implementation of this framework is not yet able to solve the hardest problems solved by dedicated state of the art methods. However, it has already shown some advantages with some very sharp BVP solution enclosures and their existence and uniqueness proof.

Tackling larger problems require strong optimization of the resolution process. For example, experiments have shown that at the beginning of the search, when intervals are large, the pruning is not efficient so reducing the order of the Taylor expansion would certainly pay off. Also, we are using today the interval integrator from the CAPD library but implementing a dedicated interval

integrator would allow integrating ODE more carefully in order to save computations that are not necessary to the CSP resolution process. Also, this interval integrator is optimized for small initial conditions, which is good for computing sharp enclosure of solutions at the end of the resolution process but which is not efficient at the beginning of the search process when domain to be proceeded are large.

Finally, some specific NCSP features like universally quantified constraints, parallelotopes domains, rigorous global optimization can naturally be used with ODE based constraints, which can lead to interesting developments.

Acknowledgments

This work was supported in part by the International Internship Program of the National Institute of Informatics, Japan. The authors are particularly grateful to Daniel Wilczak who helped them using the ODE integrator available in CAPD, and for his helpful comments on our simulations.

References

1. Raïssi, T., Ramdani, N., Candau, Y.: Set membership state and parameter estimation for systems described by nonlinear differential equations. *Automatica* 40(10), 1771–1777 (2004)
2. Granvilliers, L., Cruz, J., Barahona, P.: Parameter estimation using interval computations. *SIAM J. Sci. Comput.* 26(2), 591–612 (2005)
3. Kapela, T., Simó, C.: Computer assisted proofs for nonsymmetric planar choreographies and for stability of the eight. *Nonlinearity* 20(5), 1241 (2007)
4. Lin, Y., Stadtherr, M.A.: Guaranteed state and parameter estimation for nonlinear continuous-time systems with bounded-error measurements. *Industrial & Engineering Chemistry Research* 46(22), 7198–7207 (2007)
5. Johnson, T., Tucker, W.: Brief paper: Rigorous parameter reconstruction for differential equations with noisy data. *Automatica* 44(9), 2422–2426 (2008)
6. Lin, Y., Enszer, J.A., Stadtherr, M.A.: Enclosing all solutions of two-point boundary value problems for odes. *Computers & Chemical Engineering* 32(8), 1714–1725 (2008)
7. Cruz, J., Barahona, P.: Constraint Satisfaction Differential Problems. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 259–273. Springer, Heidelberg (2003)
8. Cruz, J., Barahona, P.: Constraint reasoning in deep biomedical models. *Artificial Intelligence in Medicine* 34(1), 77–88 (2005)
9. Moore, R.: *Interval Analysis*. Prentice-Hall, Englewood Cliffs (1966)
10. Neumaier, A.: *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge (1990)
11. Benhamou, F., Older, W.: Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming* 32(1), 1–24 (1997)
12. Van Hentenryck, P., McAllester, D., Kapur, D.: Solving polynomial systems using a branch and prune approach. *SIAM J. Numer. Anal.* 34(2), 797–827 (1997)
13. Granvilliers, L.: A symbolic-numerical branch and prune algorithm for solving nonlinear polynomial systems. *Journal of Universal Computer Science* 4(2), 125–146 (1998)

14. Goldsztejn, A., Goualard, F.: Box Consistency through Adaptive Shaving. In: Proc. of ACM SAC 2010, pp. 2049–2054 (2010)
15. Hairer, E., Nørsett, S.P., Wanner, G.: Solving Ordinary Differential Equations I. Springer, Heidelberg (2000)
16. Nedialkov, N.S., Jackson, K.R., Corliss, G.F.: Validated Solutions of Initial Value Problems for Ordinary Differential Equations. *Applied Mathematics and Computation* 105(1), 21–68 (1999)
17. Janssen, M., Deville, Y., Hentenryck, P.V.: Multistep filtering operators for ordinary differential equations. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 246–260. Springer, Heidelberg (1999)
18. Zgliczynski, P.: C^1 -Lohner Algorithm. *Foundations of Computational Mathematics* 2(4), 429–465 (2002)
19. Goldsztejn, A., Hayes, W.: Reliable Inner Approximation of the Solution Set to Initial Value Problems with Uncertain Initial Value. In: Proceedings of SCAN 2006, p. 19. IEEE Press, Los Alamitos (2006)
20. Goldsztejn, A., Michel, C., Rueher, M.: Efficient Handling of Universally Quantified Inequalities. *Constraints* 14(1), 117–135 (2008)
21. Goldsztejn, A., Granvilliers, L.: A New Framework for Sharp and Efficient Resolution of NCSP with Manifolds of Solutions. *Constraints* 15(2), 190–212 (2010)
22. Eveillard, D., Ropers, D., de Jong, H., Branlant, C., Bockmayr, A.: A multi-scale constraint programming model of alternative splicing regulation. *Theor. Comput. Sci.* 325(1), 3–24 (2004)
23. Carlson, B., Gupta, V.: Hybrid cc with interval constraints. In: HSCC 1998: Proceedings of the First International Workshop on Hybrid Systems, pp. 80–95 (1998)
24. Gupta, V., Jagadeesan, R., Saraswat, V.A.: Computing with continuous change. *Sci. Comput. Program.* 30(1-2), 3–49 (1998)
25. Kuzmic, P.: Program DYNAFIT for the Analysis of Enzyme Kinetic Data: Application to HIV Proteinase. *Analytical Biochemistry* 237(2), 260–273 (1996)
26. Mendes, P., Kell, D.: Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics* 14(10), 869–883 (1998)
27. Lin, Y., Stadtherr, M.A.: Validated solutions of initial value problems for parametric odes. *Applied Numerical Mathematics* 57(10), 1145–1162 (2007)
28. Hlaváček, V., Marek, M., Kubíček, M.: Modelling of chemical reactors – X multiple solutions of enthalpy and mass balances for a catalytic reaction within a porous catalyst particle. *Chemical Engineering Science* 23(9), 1083–1097 (1968)
29. Chen, Y.: Dynamic systems optimization. Ph. D. Thesis, University of California (2006)

Structural Tractability of Enumerating CSP Solutions

Gianluigi Greco¹ and Francesco Scarcello²

¹ Dept. of Mathematics

² DEIS, University of Calabria, 87036, Rende, Italy
ggreco@mat.unical.it, scarcello@deis.unical.it

Abstract. The problem of deciding whether CSP instances admit solutions has been deeply studied in the literature, and several structural tractability results have been derived so far. However, constraint satisfaction comes in practice as a computation problem where the focus is either on finding one solution, or on enumerating all solutions, possibly projected over some given set of output variables. The paper investigates the structural tractability of the problem of enumerating (possibly projected) solutions, where tractability means here computable with polynomial delay (WPD), since in general exponentially many solutions may be computed. A general framework based on the notion of tree projection of hypergraphs is considered, which generalizes all known decomposition methods. Tractability results have been obtained both for classes of structures where output variables are part of their specification, and for classes of structures where computability WPD must be ensured for any possible set of output variables. These results are shown to be tight, by exhibiting dichotomies for classes of structures having bounded arity and where the tree decomposition method is considered.

1 Introduction

1.1 Constraint Satisfaction and Decomposition Methods

Constraint satisfaction is often formalized as a homomorphism problem that takes as input two finite relational structures \mathbb{A} (modeling variables and scopes of the constraints) and \mathbb{B} (modeling the relations associated with constraints), and asks whether there is a homomorphism from \mathbb{A} to \mathbb{B} . Since the general problem is NP-hard, many restrictions have been considered in the literature, where the given structures have to satisfy additional conditions. In this paper, we are interested in restrictions imposed on the (usually said) left-hand structure, i.e., \mathbb{A} must be taken from some suitably defined class \mathbf{A} of structures, while \mathbb{B} is any arbitrary structure from the class “—” of all finite structures¹. Thus, we

¹ The finite property is a feature of this framework, and not a simplifying assumption. E.g., on structures with possibly infinite domains, the open question in [10] (recently answered by [15] on finite structures) would have been solved in 1993 [23].

face the so-called *uniform* constraint satisfaction problem, shortly denoted as $\text{CSP}(\mathbf{A}, -)$, where both structures are part of the input (nothing is fixed).

The decision problem $\text{CSP}(\mathbf{A}, -)$ has intensively been studied in the literature, and various classes of structures over which it can be solved in polynomial time have already been singled out (see [7,11,18,1], and the references therein). These approaches, called *decomposition methods*, are based on properties of the hypergraph $\mathcal{H}_{\mathbb{A}}$ associated with each structure $\mathbb{A} \in \mathbf{A}$. In fact, it is well-known that, for the class \mathbf{A}_a of all structures whose associated hypergraphs are acyclic, $\text{CSP}(\mathbf{A}_a, -)$ is efficiently solvable by just enforcing *generalized arc consistency* (**GAC**)—roughly, by filtering constraint relations until every pair of constraints having some variables \bar{X} in common agree on \bar{X} (that is, they have precisely the same set of allowed tuples of values on these variables \bar{X}).

Larger “islands of tractability” are then identified by generalizing hypergraph acyclicity. To this end, every decomposition method **DM** associates with any hypergraph $\mathcal{H}_{\mathbb{A}}$ some measure w of its cyclicity, called the *DM-width* of $\mathcal{H}_{\mathbb{A}}$. The tractable classes \mathbf{A} of instances (according to **DM**) are those (with hypergraphs) having bounded width, that is, whose degree of cyclicity is below some fixed threshold. For every instance \mathbb{A} in such a class \mathbf{A} and every structure \mathbb{B} , the instance (\mathbb{A}, \mathbb{B}) can be solved in polynomial-time by exploiting the solutions of a set of suitable subproblems, that we call *views*, each one solvable in polynomial-time (in fact, exponential in the—fixed—width, for all known methods). In particular, the idea is to arrange some of these views in a tree, called *decomposition*, in order to exploit the known algorithms for acyclic instances. In fact, whenever such a tree exists, instances can be solved by just enforcing **GAC** on the available views, even without computing explicitly any decomposition. This very general approach traces back to the seminal database paper [10], and it is based on the graph-theoretic notion of *tree-projection* of the pair of hypergraphs $(\mathcal{H}_{\mathbb{A}}, \mathcal{H}_{\mathcal{V}})$, associated with the input structure \mathbb{A} and with the structure \mathcal{V} of the available views, respectively (tree projections are formally defined in Section 2).

For instance, assume that the fixed threshold on the width is k : in the *generalized hypertree-width* method [13], the available views are all subproblems involving at most k constraints from the given CSP instance; in the case of *treewidth* [21], the views are all subproblems involving at most k variables; for *fractional hypertree-width*, the views are all subproblems having fractional cover-width at most k (in fact, if we require that they are computable in polynomial-time, we may instead use those subproblems defined in [19] to compute a $O(k^3)$ approximation of this notion).

Note that, for the special case of generalized hypertree-width, the fact that enforcing **GAC** on all clusters of k constraints is sufficient to solve the given instance, without computing a decomposition, has been re-derived in [5] (with proof techniques different from those in [10]). Moreover, [5] actually provided a stronger result, as it is proved that this property holds even if there is some homomorphically equivalent subproblem having generalized hypertree-width at most k . However, the corresponding *only if* result is missing in that paper, and characterizing the precise power of this **GAC** procedure for the views obtained

from all clusters of k constraints (short: k -GAC) remained an open question. For any class \mathbf{A} of instances having bounded arity (i.e., with a fixed maximum number of variables in any constraint scope of every instance of the class), the question has been answered in [2]: $\forall \mathbb{A} \in \mathbf{A}$, k -GAC is correct for every right-hand structure \mathbb{B} if, and only if, the core of \mathbb{A} has tree width at most k (recall that treewidth and generalized hypertree-width identify the same set of bounded-arity tractable classes). In its full version, the answer to this open question follows from a recent result in [15] (see Theorem 2).

In fact, for any recursively enumerable class of bounded-arity structures \mathbf{A} , it is known that this method is essentially optimal: $\text{CSP}(\mathbf{A}, -)$ is solvable in polynomial time *if, and only if*, the cores of the structures in \mathbf{A} have bounded treewidth (under standard complexity theoretic assumptions) [17]. Note that the latter condition may be equivalently stated as follows: for every $\mathbb{A} \in \mathbf{A}$ there is some \mathbb{A}' homomorphically equivalent to \mathbb{A} and such that its treewidth is below the required fixed threshold. For short, we say that such a class has bounded treewidth modulo homomorphic equivalence.

Things with unbounded-arity classes are not that clear. Generalized hypertree-width does not characterize all classes of (arbitrary) structures where $\text{CSP}(\mathbf{A}, -)$ is solvable in polynomial time [18]. It seems that a useful characterization may be obtained by relaxing the requirement that views are computable in polynomial time, and by requiring instead that such tasks are fixed-parameter tractable (FPT) [9]. In fact, towards establishing such characterization, it was recently shown in [20] that (under some reasonable technical assumptions) the problem $\text{CSP}(\mathcal{H})$, i.e., $\text{CSP}(\mathbf{A}, -)$ restricted to the instances whose associated hypergraphs belong to the class \mathcal{H} , is FPT *if, and only if*, hypergraphs in \mathcal{H} have bounded *submodular* width—a new hypergraph measure more general than *fractional* hypertree-width and, hence, than generalized hypertree-width.

It is worthwhile noting that the above mentioned tractability results for classes of instances defined modulo homomorphically equivalence are actually tractability results for the *promise* version of the problem. In fact, unless $P = NP$, there is no polynomial-time algorithm that may check whether a given instance \mathbb{A} actually belongs to such a class \mathbf{A} . In particular, it has been observed by different authors [24,4] that there are classes of instances having bounded treewidth modulo homomorphically equivalence for which answers computable in polynomial time cannot be trusted. That is, unless $P = NP$, there is no efficient way to distinguish whether a “yes” answer means that there is some solution of the problem, or that $\mathbb{A} \notin \mathbf{A}$.

In this paper, besides promise problems, we also consider the so-called *no-promise* problems, which seem more appealing for practical applications. In this case, either certified solutions are computed, or the promise $\mathbb{A} \in \mathbf{A}$ is correctly disproved. For instance, the algorithm in [5] solves the no-promise search-problem of computing a homomorphism for a given CSP instance (\mathbb{A}, \mathbb{B}) . This algorithm either computes such a homomorphism or concludes that $\mathcal{H}_{\mathbb{A}}$ has generalized hypertree-width greater than k .

1.2 Enumeration Problems

While the structural tractability of deciding whether CSP instances admit solutions has been deeply studied in the literature, the structural tractability of the corresponding computation problem received considerably less attention so far [4], though this is certainly a more appealing problem for practical applications. In particular, it is well-known that for classes of CSPs where the decision problem is tractable and a self-reduction argument applies the enumeration problem is tractable too [8,6]. Roughly, these classes have a suitable closure property such that one may fix values for the variables without going out of the class, and thus may solve the computation problem by using the (polynomial-time) algorithm for the decision problem as an oracle. In fact, for the non-uniform CSP problem, the tractability of the decision problem always entails the tractability of the search problem [6]. As observed above, this is rather different from what happens in the uniform CSP problem that we study in this paper, where this property does not hold (see [24,4], and Proposition 1), and thus a specific study for the computation problem is meaningful and necessary.

In this paper, we embark on this study, by focusing on the problem ECSP of enumerating (possibly projected) solutions. In particular, our goals are (1) to identify precisely those classes of uniform CSPs where consistency-based algorithms allow us to solve efficiently the problem and, more ambitiously, (2) to prove dichotomies saying that such classes are in fact all the tractable ones (at least for the case of bounded-arity CSPs). Since even easy instances of enumeration problems may have an exponential number of solutions, tractability means here having algorithms that compute solutions *with polynomial delay* (WPD): An algorithm M solves WPD a computation problem P if there is a polynomial $p(\cdot)$ such that, for every instance of P of size n , M discovers if there are no solutions in time $O(p(n))$; otherwise, it outputs all solutions in such a way that a new solution is computed within $O(p(n))$ time from the previous one.

Before stating our contribution, we notice that there are different facets of the enumeration problem, and thus different research directions to be explored:

(Which Decomposition Methods?) We considered the more general framework of the tree projections, where subproblems (views) may be completely arbitrary, so that our results are smoothly inherited by all (known) decomposition methods. We remark that this choice posed interesting technical challenges to our analysis, and called for solution approaches that were not explored in the earlier literature on traditional methods, such as treewidth. E.g., in this context, we cannot speak anymore of “the core” of a structure, because different isomorphic cores may have different structural properties with respect to the available views.

(Only full solutions or possibly projected solutions?) In this paper, an ECSP instance is a triple $(\mathbb{A}, \mathbb{B}, O)$, for which we have to compute all solutions (homomorphisms) projected over a set of desired output variables O , denoted by $\mathbb{A}^{\mathbb{B}}[O]$. We believe this is the more natural approach. Indeed, modeling real-world applications through CSP instances typically requires the use of “auxiliary” variables, whose precise values in the solutions are not relevant for the user, and that are (usually) filtered-out from the output. In these cases, computing all combinations

of their values occurring in solutions means wasting time, possibly exponential time. Of course, this aspect is irrelevant for the problem of computing just one solution, but is crucial for the enumeration problem.

(*Should classes of structures be aware of output variables?*) This is an important technical question. We are interested in identifying classes of tractable instances based on properties of their left-hand structures, while right-hand structures have no restrictions. What about output variables? In principle, structural properties may or may not consider the possible output variables, and in fact both approaches have been explored in the literature (see, e.g., [17]), and both approaches are dealt with in this paper. In the former output-aware case, possible output variables are suitably described in the instance structure. Unlike previous approaches that considered additional “virtual” constraints covering together all possible output variables [17], in this paper possible output variables are described as those variables X having a domain constraint $dom(X)$, that is, a distinguished unary constraint specifying the domain of this variable. Such variables are said domain restricted. In fact, this choice reflects the classical approach in constraint satisfaction systems, where variables are typically associated with domains, which are heavily exploited by constraint propagation algorithms. Note that this approach does not limit the number of solutions, while in the tractable classes considered in [17] only instances with a polynomial number of (projected) solutions may be dealt with. As far as the latter case of arbitrary sets of output variables is considered, observe that in general stronger conditions are expected to be needed for tractability. Intuitively, since we may focus on any desired substructure, no strange situations may occur, and the full instance should be really tractable.

1.3 Contribution

Output-aware classes of ECSPs:

- (1) We define a property for pairs (\mathbb{A}, O) , where \mathbb{A} is a structure and $O \subseteq A$ is a set of variables, that allows us to characterize the classes of tractable instances. Roughly, we say that (\mathbb{A}, O) is *tp-covered* through the decomposition method DM if variables in O occur in a tree projection of a certain hypergraph w.r.t. to the (hypergraph associated with the) views defined according to DM.
- (2) We describe an algorithm that solves the promise enumeration problem, by computing with polynomial delay all solutions of a given instance $(\mathbb{A}, \mathbb{B}, O)$, whenever (\mathbb{A}, O) is *tp-covered* through DM.
- (3) For the special case of (generalized hyper)tree width, we show that the above condition is also necessary for the correctness of the proposed algorithm (for every \mathbb{B}). In fact, for these traditional decomposition methods we now have a complete characterization of the power of the k -GAC approach.
- (4) For recursively enumerable classes of structures having bounded arity, we exhibit a dichotomy showing that the above tractability result is tight, for $DM = \text{treewidth}$ (and assuming $FPT \neq W[1]$).

ECSP instances over arbitrary output variables:

- (1) We describe an algorithm that, on input $(\mathbb{A}, \mathbb{B}, O)$, solves the no-promise enumeration problem. In particular, either all solutions are computed, or it infers that there exists no tree projection of $\mathcal{H}_{\mathbb{A}}$ w.r.t. $\mathcal{H}_{\mathbb{B}}$ (the hypergraph associated with the views defined according to DM). This algorithm generalizes to the tree projection framework the enumeration algorithm of projected solutions recently proposed for the special case of treewidth [4].
- (2) Finally, we give some evidence that, for bounded arity classes of instances, we cannot do better than this. In particular, having bounded width tree-decompositions of the full structure seems a necessary condition for enumerating WPD. We speak of “evidence,” instead of saying that our result completely answers the open question in [17,4], because our dichotomy theorem focuses on classes of structures satisfying the technical property of being closed under taking minors (in fact, the same property assumed in the first dichotomy result on the complexity of the decision problem on classes of graphs [16]).

2 Relational Structures and Homomorphisms

A constraint satisfaction problem may be formalized as a relational homomorphism problem. A vocabulary τ is a finite set of relation symbols of specified arities. A relational structure \mathbb{A} over τ consists of a universe A and an r -ary relation $R^{\mathbb{A}} \subseteq A^r$, for each relation symbol R in τ .

If \mathbb{A} and \mathbb{A}' are two relational structures over disjoint vocabularies, we denote by $\mathbb{A} \uplus \mathbb{A}'$ the relational structure over the (disjoint) union of their vocabularies, whose domain (resp., set of relations) is the union of those of \mathbb{A} and \mathbb{A}' .

A *homomorphism* from a relational structure \mathbb{A} to a relational structure \mathbb{B} is a mapping $h : A \mapsto B$ such that, for every relation symbol $R \in \mathbb{A}$, and for every tuple $\langle a_1, \dots, a_r \rangle \in R^{\mathbb{A}}$, it holds that $\langle h(a_1), \dots, h(a_r) \rangle \in R^{\mathbb{B}}$. For any set $X \subseteq A$, denote by $h[X]$ the restriction of h to X . The set of all possible homomorphisms from \mathbb{A} to \mathbb{B} is denoted by $\mathbb{A}^{\mathbb{B}}$, while $\mathbb{A}^{\mathbb{B}}[X]$ denotes the set of their restrictions to X .

An instance of the constraint satisfaction problem (CSP) is a pair (\mathbb{A}, \mathbb{B}) where \mathbb{A} is called a *left-hand structure* (short: ℓ -structure) and \mathbb{B} is called a *right-hand structure* (short: r -structure). In the classical decision problem, we have to decide whether there is a homomorphism from \mathbb{A} to \mathbb{B} , i.e., whether $\mathbb{A}^{\mathbb{B}} \neq \emptyset$. In an instance of the corresponding enumeration problem (denoted by ECSP) we are additionally given a set of *output* elements $O \subseteq A$; thus, an instance has the form $(\mathbb{A}, \mathbb{B}, O)$. The goal is to compute the restrictions to O of all possible homomorphisms from \mathbb{A} to \mathbb{B} , i.e., $\mathbb{A}^{\mathbb{B}}[O]$. If $O = \emptyset$, the computation problem degenerates to the decision one. Formally, let $h_{\emptyset} : \emptyset \mapsto \text{true}$ denote (the constant mapping to) the Boolean value *true*; then, define $\mathbb{A}^{\mathbb{B}}[\emptyset] = \{h_{\emptyset}\}$ (resp., $\mathbb{A}^{\mathbb{B}}[\emptyset] = \emptyset$) if there is some (resp., there is no) homomorphism from \mathbb{A} to \mathbb{B} .

In the constraint satisfaction jargon, the elements of A (the domain of the ℓ -structure \mathbb{A}) are the variables, and there is a constraint $C = (\langle a_1 \dots, a_r \rangle, R^{\mathbb{B}})$

for every tuple $\langle a_1 \dots, a_r \rangle \in R^A$ and every relation symbol $R \in \tau$. The tuple of variables is usually called the scope of C , while $R^{\mathbb{B}}$ is called the relation of C . Any homomorphism from \mathbb{A} to \mathbb{B} is thus a mapping from the variables in A to the elements in B (often called domain values) that satisfies all constraints, and it is also called a solution (or a projected solution, if it is restricted to a subset of the variables).

Two relational structures \mathbb{A} and A' are homomorphically equivalent if there is a homomorphism from \mathbb{A} to A' and vice-versa. A structure A' is a substructure of \mathbb{A} if $A' \subseteq A$ and $R^{A'} \subseteq R^A$, for each symbol $R \in \tau$. Moreover, A' is a *core* of \mathbb{A} if it is a substructure of \mathbb{A} such that: (1) there is a homomorphism from \mathbb{A} to A' , and (2) there is no substructure A'' of A' , with $A'' \neq A'$, satisfying (1).

3 Decomposition Methods, Views, and Tree Projections

Throughout the following sections we assume that (\mathbb{A}, \mathbb{B}) is a given connected CSP instance, and we shall seek to compute its solutions (possibly restricted over a desired set of output variables) by combining the solutions of suitable sets of subproblems, available as additional distinguished constraints called *views*.

Let $\mathbb{A}_{\mathcal{V}}$ be an ℓ -structure with the same domain as \mathbb{A} . We say that $\mathbb{A}_{\mathcal{V}}$ is a *view structure* (short: *v-structure*) if

- its vocabulary $\tau_{\mathcal{V}}$ is disjoint from the vocabulary τ of \mathbb{A} ;
- every relation $R^{A_{\mathcal{V}}}$ contains a single tuple whose variables will be denoted by $var(R^{A_{\mathcal{V}}})$. That is, there is a one-to-one correspondence between views and relation symbols in $\tau_{\mathcal{V}}$, so that we shall often use the two terms interchangeably;
- for every relation $R \in \tau$ and every tuple $t \in R^A$, there is some relation $R_t \in \tau_{\mathcal{V}}$, called *base view*, such that $\{t\} = R_t^{A_{\mathcal{V}}}$, i.e., for every constraint in \mathbb{A} there is a corresponding view in $\mathbb{A}_{\mathcal{V}}$.

Let $\mathbb{B}_{\mathcal{V}}$ be an r -structure. We say that $\mathbb{B}_{\mathcal{V}}$ is *legal* (w.r.t. $\mathbb{A}_{\mathcal{V}}$ and (\mathbb{A}, \mathbb{B})) if

- its vocabulary is $\tau_{\mathcal{V}}$;
- For every view $R \in \tau_{\mathcal{V}}$, $R^{\mathbb{B}_{\mathcal{V}}} \supseteq \mathbb{A}^{\mathbb{B}}[w]$ holds, where $w = var(R^{A_{\mathcal{V}}})$. That is, every subproblem is not more restrictive than the full problem.
- For every base view $R_t \in \tau_{\mathcal{V}}$, $R_t^{\mathbb{B}_{\mathcal{V}}} \subseteq R^{\mathbb{B}}$. That is, any base view is at least as restrictive as the “original” constraint associated with it.

The following fact immediately follows from the above properties.

Fact 1. *Let $\mathbb{B}_{\mathcal{V}}$ be any r -structure that is legal w.r.t. $\mathbb{A}_{\mathcal{V}}$ and (\mathbb{A}, \mathbb{B}) . Then, $\forall O \subseteq A$, the ECSP instance $(\mathbb{A}_{\mathcal{V}}, \mathbb{B}_{\mathcal{V}}, O)$ has the same set of solutions as $(\mathbb{A}, \mathbb{B}, O)$.*

In fact, all structural decomposition methods define some way to build the views to be exploited for solving the given CSP instance. In our framework, we associate with any decomposition method DM a pair of polynomial-time computable functions ℓ -DM and r -DM that, given any CSP instance (\mathbb{A}, \mathbb{B}) , compute the pair $(\mathbb{A}_{\mathcal{V}}, \mathbb{B}_{\mathcal{V}})$, where $\mathbb{A}_{\mathcal{V}} = \ell\text{-DM}(\mathbb{A})$ is a *v-structure*, and $\mathbb{B}_{\mathcal{V}} = r\text{-DM}(\mathbb{A}, \mathbb{B})$

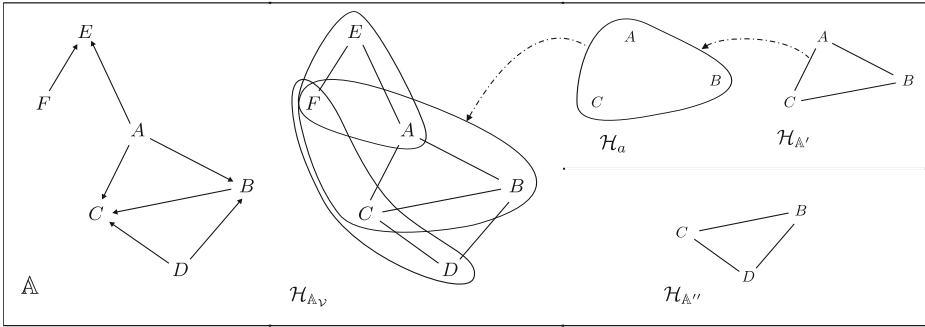


Fig. 1. A structure \mathbb{A} . A hypergraph $\mathcal{H}_{\mathbb{A}_v}$ such that $(\mathcal{H}_{\mathbb{A}}, \mathcal{H}_{\mathbb{A}_v})$ has no tree projections. Two hypergraphs $\mathcal{H}_{\mathbb{A}'}$ and $\mathcal{H}_{\mathbb{A}''}$, where \mathbb{A}' and \mathbb{A}'' are cores of \mathbb{A} . A tree projection \mathcal{H}_{α} of $(\mathcal{H}_{\mathbb{A}'}, \mathcal{H}_{\mathbb{A}_v})$.

is a legal r -structure² For instance, for any fixed natural number k , the *generalized hypertree decomposition method* [12] (short: hw_k) is associated with the functions $\ell\text{-}hw_k$ and $r\text{-}hw_k$ that, given a CSP instance (\mathbb{A}, \mathbb{B}) , build the pair $(\ell\text{-}hw_k(\mathbb{A}), r\text{-}hw_k(\mathbb{A}, \mathbb{B}))$ where, for each subset C of at most k constraints from (\mathbb{A}, \mathbb{B}) , there is a view R_C such that: (1) $\text{var}(R_C^{\ell\text{-}hw_k(\mathbb{A})})$ is the set of all variables occurring in C , and (2) the tuples in $R_C^{r\text{-}hw_k(\mathbb{A}, \mathbb{B})}$ are the solutions of the subproblem encoded by C . Similarly, the *tree decomposition method* [21] (tw_k) is defined as above, but we consider each subset of at most k variables in \mathbb{A} instead of each subset of at most k constraints.

3.1 Tree Projections for CSP Instances

In this paper we are interested in restrictions imposed on left-hand structures of CSP instances, based on some decomposition method DM. To this end, we associate with any ℓ -structure \mathbb{A} a hypergraph $\mathcal{H}_{\mathbb{A}} = (A, H)$, whose set of nodes is equal to the set of variables A and where, for each constraint scope in $R^{\mathbb{A}}$, the set H of hyperedges contains a hyperedge including all its variables (no further hyperedge is in H). In particular, the v -structure $\mathbb{A}_v = \ell\text{-DM}(\mathbb{A})$ is associated with a hypergraph $\mathcal{H}_{\mathbb{A}_v} = (A, H)$, whose set of nodes is the set of variables A and where, for each view $R \in \tau_v$, the set H contains the hyperedge $\text{var}(R^{\mathbb{A}_v})$. In the following, for any hypergraph \mathcal{H} , we denote its nodes and edges by $\text{nodes}(\mathcal{H})$ and $\text{edges}(\mathcal{H})$, respectively.

Example 1. Consider the ℓ -structure \mathbb{A} whose vocabulary just contains the binary relation symbol R , and such that $R^{\mathbb{A}} = \{\langle F, E \rangle, \langle A, E \rangle, \langle A, C \rangle, \langle A, B \rangle, \langle B, C \rangle, \langle D, B \rangle, \langle D, C \rangle\}$. Such a simple one-binary-relation structure may be

² A natural extension of this notion may be to consider FPT decomposition methods, where functions $\ell\text{-DM}$ and $r\text{-DM}$ are computable in fixed-parameter polynomial-time. For the sake of presentation and space, we do not consider FPT decomposition methods in this paper, but our results can be extended to them rather easily.

easily represented by the directed graph in the left part of Figure 11, where edge orientation reflects the position of the variables in R . In this example, the associated hypergraph $\mathcal{H}_{\mathbb{A}}$ is just the undirected version of this graph. Let DM be a method that, on input \mathbb{A} , builds the v -structure $\mathbb{A}_v = \ell\text{-DM}(\mathbb{A})$ consisting of the seven base views of the form R_t , for each tuple $t \in R^{\mathbb{A}}$, plus the three relations R_1, R_2 , and R_3 such that $R_1^{\mathbb{A}_v} = \{\langle A, E, F \rangle\}$, $R_2^{\mathbb{A}_v} = \{\langle A, B, C, F \rangle\}$, and $R_3^{\mathbb{A}_v} = \{\langle C, D, F \rangle\}$. Figure 11 also reports $\mathcal{H}_{\mathbb{A}_v}$. \triangleleft

A hypergraph \mathcal{H} is *acyclic* iff it has a *join tree* [3], i.e., a tree $JT(\mathcal{H})$, whose vertices are the hyperedges of \mathcal{H} , such that if a node X occurs in two hyperedges h_1 and h_2 of \mathcal{H} , then h_1 and h_2 are connected in $JT(\mathcal{H})$, and X occurs in each vertex on the unique path linking h_1 and h_2 in $JT(\mathcal{H})$.

For two hypergraphs \mathcal{H}_1 and \mathcal{H}_2 , we write $\mathcal{H}_1 \leq \mathcal{H}_2$ iff each hyperedge of \mathcal{H}_1 is contained in at least one hyperedge of \mathcal{H}_2 . Let $\mathcal{H}_1 \leq \mathcal{H}_2$. Then, a *tree projection* of \mathcal{H}_1 with respect to \mathcal{H}_2 is an acyclic hypergraph \mathcal{H}_a such that $\mathcal{H}_1 \leq \mathcal{H}_a \leq \mathcal{H}_2$. Whenever such a hypergraph \mathcal{H}_a exists, we say that the pair $(\mathcal{H}_1, \mathcal{H}_2)$ has a tree projection (also, we say that \mathcal{H}_1 has a tree projection w.r.t. \mathcal{H}_2). The problem of deciding whether a pair of hypergraphs has a tree projection is called the *tree projection problem*, and it has recently been proven to be NP-complete [14].

Example 2. Consider again the setting of Example 1. It is immediate to check that the pair of hypergraphs $(\mathcal{H}_{\mathbb{A}}, \mathcal{H}_{\mathbb{A}_v})$ does not have any tree projection. Consider instead the (hyper)graph $\mathcal{H}_{\mathbb{A}'}$ reported on the right of Figure 11. The acyclic hypergraph \mathcal{H}_a is a tree projection of $\mathcal{H}_{\mathbb{A}'}$ w.r.t. $\mathcal{H}_{\mathbb{A}_v}$. In particular, note that the hyperedge $\{A, B, C\} \in \text{edges}(\mathcal{H}_a)$ “absorbs” the cycle in $\mathcal{H}_{\mathbb{A}'}$, and that $\{A, B, C\}$ is in its turn contained in the hyperedge $\{A, B, C, F\} \in \text{edges}(\mathcal{H}_{\mathbb{A}_v})$. \triangleleft

Note that all the (known) structural decomposition methods can be recast as special cases of tree projections, since they just differ in how they define the set of views to be built for evaluating the CSP instance. For instance, a hypergraph $\mathcal{H}_{\mathbb{A}}$ has generalized hypertree width (resp., treewidth) at most k if and only if there is a tree projection of $\mathcal{H}_{\mathbb{A}}$ w.r.t. $\mathcal{H}_{\ell\text{-}hw_k(\mathbb{A})}$ (resp., w.r.t. $\mathcal{H}_{\ell\text{-}tw_k(\mathbb{A})}$).

However, the setting of tree projections is more general than such traditional decomposition approaches, as it allows us to consider arbitrary sets of views, which often require more care and different techniques. As an example, we shall illustrate below that in the setting of tree projections it does not make sense to talk about “the” core of an ℓ -structure, because different isomorphic cores may differently behave with respect to the available views. This phenomenon does not occur, e.g., for generalized hypertree decompositions, where all combinations of k constraints are available as views.

Example 3. Consider the structure \mathbb{A} illustrated in Example 1, and the structures \mathbb{A}' and \mathbb{A}'' over the same vocabulary as \mathbb{A} , and such that $R^{\mathbb{A}'} = \{\langle A, C \rangle, \langle A, B \rangle, \langle B, C \rangle\}$ and $R^{\mathbb{A}''} = \{\langle B, C \rangle, \langle D, B \rangle, \langle D, C \rangle\}$. The hypergraphs $\mathcal{H}_{\mathbb{A}'}$ and $\mathcal{H}_{\mathbb{A}''}$ are reported in Figure 11. Note that \mathbb{A}' and \mathbb{A}'' are two (isomorphic) cores of \mathbb{A} , but they have completely different structural properties. Indeed, $(\mathcal{H}_{\mathbb{A}'}, \mathcal{H}_{\mathbb{A}_v})$ admits a tree projection (recall Example 2), while $(\mathcal{H}_{\mathbb{A}''}, \mathcal{H}_{\mathbb{A}_v})$ does not. \triangleleft

3.2 CSP Instances and tp-Coverings

We complete the picture of our unifying framework to deal with decomposition methods for constraint satisfaction problems, by illustrating some recent results in [15], which will be useful to our ends.

For a set of variables $O = \{X_1, \dots, X_r\}$, let \mathbb{S}_O denote the structure with a fresh r -ary relation symbol R_O and domain O , such that $R_O^{\mathbb{A}_O} = \{\langle X_1, \dots, X_r \rangle\}$.

Definition 1. Let $\mathbb{A}_\mathcal{V}$ be a v -structure. A set of variables $O \subseteq A$ is *tp-covered* in $\mathbb{A}_\mathcal{V}$ if there is a core \mathbb{A}' of $\mathbb{A} \uplus \mathbb{S}_O$ such that $(\mathcal{H}_{\mathbb{A}'}, \mathcal{H}_{\mathbb{A}_\mathcal{V}})$ has a tree projection. \square

For instance, the variables $\{A, B, C\}$ are *tp-covered* in the v -structure $\mathbb{A}_\mathcal{V}$ discussed in Example 1. In particular, note that the structure $\mathbb{A} \uplus \mathbb{S}_{\{A, B, C\}}$ is associated with the same hypergraph $\mathcal{H}_{\mathbb{A}'}$ that has a tree projection w.r.t. $\mathcal{H}_{\mathbb{A}_\mathcal{V}}$ (cf. Example 3). Instead, the variables $\{B, C, D\}$ are not *tp-covered* in $\mathbb{A}_\mathcal{V}$.

Given a CSP instance $(\mathbb{A}_\mathcal{V}, \mathbb{B}_\mathcal{V})$, we denote by $\text{GAC}(\mathbb{A}_\mathcal{V}, \mathbb{B}_\mathcal{V})$ the r -structure that is obtained by enforcing generalized arc consistency on $(\mathbb{A}_\mathcal{V}, \mathbb{B}_\mathcal{V})$. The precise relationship between generalized-arc-consistent views and *tp-covered* sets of variables has recently been stated in [15], thus answering a long standing open question [10,23] about the relationship between the existence of tree projections and (local and global) consistency properties. As a consequence of this result, we characterize the power of local-consistency for any decomposition method DM such that, for each pair (\mathbb{A}, \mathbb{B}) , each view in $\mathbb{B}_\mathcal{V} = r\text{-DM}(\mathbb{A}, \mathbb{B})$ contains the solutions of the subproblem encoded by the constraints over which it is defined. For the sake of simplicity, we state below the result specialized to the well-known decomposition methods tw_k and hw_k .⁴

Theorem 2. Let DM be a decomposition method in $\{tw_k, hw_k\}$, let \mathbb{A} be an ℓ -structure, and let $\mathbb{A}_\mathcal{V} = \ell\text{-DM}(\mathbb{A})$. The following are equivalent:

- (1) A set of variables $O \subseteq A$ is *tp-covered* in $\mathbb{A}_\mathcal{V}$;
- (2) For every r -structure \mathbb{B} , and for every relation $R \in \tau_\mathcal{V}$ with $O \subseteq \text{var}(R^{\mathbb{A}_\mathcal{V}})$, $R^{\text{GAC}(\mathbb{A}_\mathcal{V}, \mathbb{B}_\mathcal{V})}[O] = \mathbb{A}^\mathbb{B}[O]$, where $\mathbb{B}_\mathcal{V} = r\text{-DM}(\mathbb{A}, \mathbb{B})$.

Note that, for the case of treewidth, we re-derive the nice result of [2], while for generalized hypertree-width we obtain the corresponding answer for the unbounded-arity case. In words, the result states that enforcing generalized arc consistency on the available views is a sound and complete procedure to solve ECSP instances *if, and only if*, we are interested in (projected) solutions over output variables that are *tp-covered* and occur together in some available view. Thus, in these cases, all solutions can be computed in polynomial time.

The more general case where output variables are arbitrary (i.e., not necessarily included in some available view) is explored in the rest of this paper.

³ For the sake of completeness, note that we use here a core \mathbb{A}' because we found it more convenient for the presentation and the proofs. However, it is straightforward to check that this notion can be equivalently stated in terms of any structure homomorphically equivalent to $\mathbb{A} \uplus \mathbb{S}_O$. The same holds for the related Definition 3.

⁴ All proofs of this paper are available in the CoRR report 1005.1567 (at arxiv.org).

4 Enumerating Solutions of Output-Aware CSP Instances

The goal of this section is to study the problem of enumerating CSP solutions for classes of instances where possible output variables are part of the structure of the given instance. This is formalized by assuming that the relational structure contains *domain constraints* that specify the domains for such variables.

Definition 2. A variable $X \in A$ is *domain restricted* in the ℓ -structure \mathbb{A} if there is a unary distinguished (domain) relation symbol $dom(X) \in \tau$ with $\{\langle X \rangle\} = dom(X)^{\mathbb{A}}$. The set of all domain restricted variables is denoted by $dru(\mathbb{A})$. \square

We say that an ECSP instance $(\mathbb{A}, \mathbb{B}, O)$ is domain restricted if $O \subseteq dru(\mathbb{A})$. Of course, if it is not, then one may easily build in linear time an equivalent domain-restricted ECSP instance where an additional fresh unary constraint is added for every output variable, whose values are taken from any constraint relation where that variable occurs. We say that such an instance is a domain-restricted version of $(\mathbb{A}, \mathbb{B}, O)$.

Figure 2 shows an algorithm, named `ComputeAllSolutionsDM`, that computes the solutions of a given ECSP instance. The algorithm is parametric w.r.t. any chosen decomposition method DM , and works as follows. Firstly, `ComputeAllSolutionsDM` starts by transforming the instance $(\mathbb{A}, \mathbb{B}, O)$ into a domain restricted one, and by constructing the views in $(\mathbb{A}_{\mathcal{V}}, \mathbb{B}_{\mathcal{V}})$ via DM . Then, it invokes the procedure `Propagate`. This procedure backtracks over the output variables $\{X_1, \dots, X_m\}$: At each step i , it tries to assign a value to X_i from its domain view⁵ and defines this value as the unique one available in that domain, in order to “propagate” such an assignment over all other views. This is accomplished by enforcing generalized arc-consistency each time the procedure is invoked. Eventually, whenever an assignment is computed for all the variables in O , this solution is returned in output, and the algorithm proceeds by backtracking again trying different values.

4.1 Tight Characterizations

To characterize the correctness of `ComputeAllSolutionsDM`, we need to define a structural property that is related to the one stated in Definition 1. Below, differently from Definition 1 where the set of output variables O is treated as a whole, each variable in O has to be tp-covered as a singleton set.

Definition 3. Let $(\mathbb{A}, \mathbb{B}, O)$ be an ECSP instance. We say that (\mathbb{A}, O) is *tp-covered* through DM if there is a core \mathbb{A}' of $\mathbb{A} \uplus \bigsqcup_{X \in O} \mathbb{S}_{\{X\}}$ such that $(\mathcal{H}_{\mathbb{A}'}, \mathcal{H}_{\ell\text{-DM}(\mathbb{A})})$ has a tree projection. \square

Note that the above definition is purely structural, because (the right-hand structure) \mathbb{B} plays no role there. In fact, the following result states that this definition captures classes of instances where `ComputeAllSolutionsDM` is correct.

⁵ In the algorithm we denote by $dom(X)$ the base view in $\tau_{\mathcal{V}}$ associated with the input constraint $dom(X) \in \tau$ —no confusion arises as the algorithm only works on views.

<p>Input: An ECSP instance $(\mathbb{A}, \mathbb{B}, O)$, where $O = \{X_1, \dots, X_m\}$;</p> <p>Output: $\mathbb{A}^{\mathbb{B}}[O]$;</p> <p>Method: update $(\mathbb{A}, \mathbb{B}, O)$ with any of its domain-restricted versions; let $\mathbb{A}_V := \ell\text{-DM}(\mathbb{A})$, $\mathbb{B}_V := r\text{-DM}(\mathbb{A}, \mathbb{B})$; invoke Propagate$(1, (\mathbb{A}_V, \mathbb{B}_V), m, \langle \rangle)$;</p> <hr/> <p>Procedure Propagate$(i$: integer, $(\mathbb{A}_V, \mathbb{B}_V)$: pair of structures, m: integer, $\langle a_1, \dots, a_{i-1} \rangle$: tuple of values in A^i);</p> <p>begin</p> <ol style="list-style-type: none"> 1. let $\mathbb{B}'_V := \text{GAC}(\mathbb{A}_V, \mathbb{B}_V)$; 2. let $\text{activeValues} := \text{dom}(X_i)^{\mathbb{B}'_V}$; 3. for each element $\langle a_i \rangle \in \text{activeValues}$ do 4. if $i = m$ then 5. output $\langle a_1, \dots, a_{m-1}, a_m \rangle$; 6. else 7. update $\text{dom}(X_i)^{\mathbb{B}'_V}$ with $\{\langle a_i \rangle\}$; /* X_i is fixed to value a_i */ 8. Propagate$(i + 1, (\mathbb{A}_V, \mathbb{B}'_V), m, \langle a_1, \dots, a_{i-1}, a_i \rangle)$; <p>end.</p>

Fig. 2. Algorithm $\text{ComputeAllSolutions}_{\text{DM}}$

Theorem 3. Let DM be a decomposition method, let \mathbb{A} be an ℓ -structure, and let $O \subseteq A$ be a set of variables. Assume that (\mathbb{A}, O) is tp -covered through DM . Then, for every r -structure \mathbb{B} , $\text{ComputeAllSolutions}_{\text{DM}}$ computes the set $\mathbb{A}^{\mathbb{B}}[O]$.

We now complete the picture by observing that Definition 3 also provides the necessary conditions for the correctness of $\text{ComputeAllSolutions}_{\text{DM}}$. As in Theorem 2, we state below the result specialized to the methods tw_k and hw_k .

Theorem 4. Let DM be a decomposition method in $\{tw_k, hw_k\}$, let \mathbb{A} be an ℓ -structure, and let $O \subseteq A$ be a set of variables. If for every r -structure \mathbb{B} $\text{ComputeAllSolutions}_{\text{DM}}$ computes $\mathbb{A}^{\mathbb{B}}[O]$, then (\mathbb{A}, O) is tp -covered through DM .

We next analyze the complexity of $\text{ComputeAllSolutions}_{\text{DM}}$.

Theorem 5. Let \mathbb{A} be an ℓ -structure, and $O \subseteq A$ be a set of variables. If (\mathbb{A}, O) is tp -covered through DM , then $\text{ComputeAllSolutions}_{\text{DM}}$ runs WPD.

By the above theorem and the definition of domain restricted variables, the following can easily be established.

Corollary 1. Let \mathbf{A} be any class of ℓ -structures such that, for each $\mathbb{A} \in \mathbf{A}$, $(\mathbb{A}, \text{drv}(\mathbb{A}))$ is tp -covered through DM . Then, for every r -structure \mathbb{B} , and for every set of variables $O \subseteq \text{drv}(\mathbb{A})$, the ECSP instance $(\mathbb{A}, \mathbb{B}, O)$ is solvable WPD.

In the case of bounded arity structures and if the (hyper)tree width is the chosen decomposition method, it is not hard to see that the result in Corollary 1 is essentially tight. Indeed, the implication $(2) \Rightarrow (1)$ in the theorem below easily follows from the well-known dichotomy for the decision version [17], which is obtained in the special case of ECSP instances without output variables ($O = \emptyset$).

Theorem 6. *Assume $FPT \neq W[1]$. Let \mathbf{A} be any class of ℓ -structures of bounded arity. Then, the following are equivalent:*

- (1) \mathbf{A} has bounded treewidth modulo homomorphic equivalence;
- (2) For every $\mathbb{A} \in \mathbf{A}$, for every r -structure \mathbb{B} , and for every set of variables $O \subseteq \text{drv}(\mathbb{A})$, the ECSP instance $(\mathbb{A}, \mathbb{B}, O)$ is solvable WPD.

Actually, from an application perspective of this result, we observe that there is no efficient algorithm for the no-promise problem for such classes. In fact, the following proposition formalizes and generalizes previous observations from different authors about the impossibility of actually trusting positive answers in the (promise) decision problem [24,4].

We say that a pair (h, c) is a certified projected solution of $(\mathbb{A}, \mathbb{B}, O)$ if, by using the certificate c , one may check in polynomial-time (w.r.t. the size of $(\mathbb{A}, \mathbb{B}, O)$) whether $h \in \mathbb{A}^{\mathbb{B}}[O]$. E.g., any full solution extending h is clearly such a certificate. If $O = \emptyset$, h is also empty, and c is intended to be a certificate that (\mathbb{A}, \mathbb{B}) is a “Yes” instance of the decision CSP. Finally, we assume that the empty output is always a certified answer, in that it entails that the input is a “No” instance, without the need for an explicit certificate of this property.

Proposition 1. *The following problem is NP-hard: Given any ECSP instance $(\mathbb{A}, \mathbb{B}, O)$, compute a certified solution in $\mathbb{A}^{\mathbb{B}}[O]$, whenever (\mathbb{A}, O) is tp-covered through DM; otherwise, there are no requirements and any output is acceptable. Hardness holds even if DM is the treewidth method with $k = 2$, the vocabulary contains just one binary relation symbol, and $O = \emptyset$.*

5 Enumeration over Arbitrary Output Variables

In this section we consider structural properties that are independent of output variables, so that tractability must hold for any desired sets of output variables. For this case, we are able to provide certified solutions WPD, which seems the more interesting notion of tractability for actual applications.

Figure 3 shows the `ComputeCertifiedSolutionsDM` algorithm computing all solutions of an ECSP instance, with a certificate for each of them. The algorithm is parametric w.r.t. any chosen decomposition method DM, and resembles in its structure the `ComputeAllSolutionsDM` algorithm. The main difference is that, after having found an assignment $\langle a_1, \dots, a_m \rangle$ for the variables in O , `ComputeCertifiedSolutionsDM` still iterates over the remaining variables in order to find a certificate for that projected solution. Of course, `ComputeCertifiedSolutionsDM` does not backtrack over the possible values to be assigned to the variables in $\{X_{m+1}, \dots, X_n\}$, since just one extension suffices to certify that this partial solution can be extended to a full one. Thus, we break the cycle after an element $\langle a_i \rangle$ is picked from its domain and correctly propagated, for each $i > m$, so that in these cases we eventually backtrack directly to $i = m$ (to look for a new projected solution).

<p>Input: An ECSP instance $(\mathbb{A}, \mathbb{B}, O)$, where $O = \{X_1, \dots, X_m\}$; Output: for each solution $h \in \mathbb{A}^{\mathbb{B}}[O]$, a certified solution (h, h'); Method: let $A = \{X_1, \dots, X_m, X_{m+1}, \dots, X_n\}$ be the variables of \mathbb{A}; update $(\mathbb{A}, \mathbb{B}, A)$ with any of its domain restricted versions; let $\mathbb{A}_V := \ell\text{-DM}(\mathbb{A})$, $\mathbb{B}_V := r\text{-DM}(\mathbb{A}, \mathbb{B})$; invoke $\text{CPropagate}(1, (\mathbb{A}_V, \mathbb{B}_V), m, \langle \rangle)$;</p> <hr style="border: none; border-top: 1px solid black; margin: 5px 0;"/> <p>Procedure CPropagate(i: integer, $(\mathbb{A}_V, \mathbb{B}_V)$: pair of structures, m: integer, (a_1, \dots, a_{i-1}): tuple of values in A^i);</p> <p>begin</p> <ol style="list-style-type: none"> 1. let $\mathbb{B}'_V := \text{GAC}(\mathbb{A}_V, \mathbb{B}_V)$; 2. if $i > 1$ and \mathbb{B}'_V is empty then output “DM failure” and HALT; 3. let $\text{activeValues} := \text{dom}(X_i)^{\mathbb{B}'_V}$; 4. for each element $\langle a_i \rangle \in \text{activeValues}$ do 5. if $i = n$ then 6. output the certified solution $(\langle a_1, \dots, a_m \rangle, \langle a_{m+1}, \dots, a_n \rangle)$; 7. else 8. update $\text{dom}(X_i)^{\mathbb{B}'_V}$ with $\{\langle a_i \rangle\}$; /* X_i is fixed to value a_i */ 9. CPropagate($i + 1, (\mathbb{A}_V, \mathbb{B}'_V), m, \langle a_1, \dots, a_{i-1}, a_i \rangle$); 10. if $i > m$ then BREAK; <p>end.</p>

Fig. 3. Algorithm $\text{ComputeCertifiedSolutions}_{\text{DM}}$

Note that $\text{ComputeCertifiedSolutions}_{\text{DM}}$ incrementally outputs various solutions, but it halts the computation if the current r -structure \mathbb{B}'_V becomes empty. As an important property of the algorithm, even when this abnormal exit condition occurs, we are guaranteed that all the elements provided as output until this event are indeed solutions. Moreover, if no abnormal termination occurs, then we are guaranteed that all solutions will actually be computed. Correctness follows easily from the same arguments used for $\text{ComputeAllSolutions}_{\text{DM}}$, by observing that, whenever $(\mathcal{H}_{\mathbb{A}}, \mathcal{H}_{\ell\text{-DM}(\mathbb{A})})$ has a tree projection, the full set of variables A is tp -covered through DM.

Theorem 7. *Let \mathbb{A} be an ℓ -structure, and $O \subseteq A$ be a set of variables. Then, for every r -structure \mathbb{B} , $\text{ComputeCertifiedSolutions}_{\text{DM}}$ computes WPD a subset of the solutions in $\mathbb{A}^{\mathbb{B}}[O]$, with a certificate for each of them. Moreover,*

- *If $\text{ComputeCertifiedSolutions}_{\text{DM}}$ outputs “DM failure”, then $(\mathcal{H}_{\mathbb{A}}, \mathcal{H}_{\ell\text{-DM}(\mathbb{A})})$ does not have a tree projection;*
- *otherwise, $\text{ComputeCertifiedSolutions}_{\text{DM}}$ computes WPD $\mathbb{A}^{\mathbb{B}}[O]$.*

Moreover, we next give some evidence that, for bounded arity classes of instances, we cannot do better than this. In particular, having bounded width tree-decompositions of the full structure seems a necessary condition for the tractability of the enumeration problem WPD w.r.t. arbitrary sets of output variables (and for every r -structure).

Theorem 8. *Assume $\text{FPT} \neq \text{W}[1]$. Let \mathbf{A} be any bounded-arity recursively-enumerable class of ℓ -structures closed under taking minors. Then, the following are equivalent:*

- (1) \mathbf{A} has bounded treewidth;
 (2) For every $\mathbb{A} \in \mathbf{A}$, for every r -structure \mathbb{B} , and for every set of variables $O \subseteq A$, the ECSP instance $(\mathbb{A}, \mathbb{B}, O)$ is solvable WPD.

References

1. Adler, I.: Tree-Related Widths of Graphs and Hypergraphs. *SIAM Journal Discrete Mathematics* 22(1), 102–123 (2008)
2. Atserias, A., Bulatov, A., Dalmau, V.: On the Power of k -Consistency. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) *ICALP 2007*. LNCS, vol. 4596, pp. 279–290. Springer, Heidelberg (2007)
3. Bernstein, P.A., Goodman, N.: The power of natural semijoins. *SIAM Journal on Computing* 10(4), 751–771 (1981)
4. Bulatov, A., Dalmau, V., Grohe, M., Marx, D.: Enumerating Homomorphism. In: *Proc. of STACS 2009*, pp. 231–242 (2009)
5. Chen, H., Dalmau, V.: Beyond Hypertree Width: Decomposition Methods Without Decompositions. In: van Beek, P. (ed.) *CP 2005*. LNCS, vol. 3709, pp. 167–181. Springer, Heidelberg (2005)
6. Cohen, D.A.: Tractable Decision for a Constraint Language Implies Tractable Search. *Constraints* 9(3), 219–229 (2004)
7. Cohen, D., Jeavons, P., Gyssens, M.: A unified theory of structural tractability for constraint satisfaction problems. *Journal of Computer and System Sciences* 74(5), 721–743 (2008)
8. Dechter, R., Itai, A.: Finding All Solutions if You can Find One. In: *AAAI 1992 Workshop on Tractable Reasoning*, pp. 35–39 (1992)
9. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, New York (1999)
10. Goodman, N., Shmueli, O.: The tree projection theorem and relational query processing. *Journal of Computer and System Sciences* 29(3), 767–786 (1984)
11. Gottlob, G., Leone, N., Scarcello, F.: A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence* 124(2), 243–282 (2000)
12. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences* 64(3), 579–627 (2002)
13. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. of Computer and System Sciences* 66(4), 775–808 (2003)
14. Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: NP-hardness and tractable variants. *Journal of the ACM* 56(6) (2009)
15. Greco, G., Scarcello, F.: The Power of Tree Projections: Local Consistency, Greedy Algorithms, and Larger Islands of Tractability. In: *Proc. of PODS 2010*, pp. 327–338 (2010)
16. Grohe, M., Schwentick, T., Segoufin, L.: When is the evaluation of conjunctive queries tractable? In: *Proc. of STOC 2001*, pp. 657–666 (2001)
17. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM* 54(1) (2007)
18. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: *Proc. of SODA 2006*, pp. 289–298 (2006)
19. Marx, D.: Approximating fractional hypertree width. In: *Proc. of SODA 2009*, pp. 902–911 (2008)

20. Marx, D.: Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. In: Proc. of STOC 2010, pp. 735–744 (2010)
21. Robertson, N., Seymour, P.D.: Graph minors III: Planar tree-width. *Journal of Combinatorial Theory, Series B* 36, 49–64 (1984)
22. Robertson, N., Seymour, P.D.: Graph minors V: Excluding a planar graph. *Journal of Combinatorial Theory, Series B* 41, 92–114 (1986)
23. Sagiv, Y., Shmueli, O.: O Shmueli. Solving Queries by Tree Projections. *ACM Transaction on Database Systems* 18(3), 487–511 (1993)
24. Scarcello, F., Gottlob, G., Greco, G.: Uniform Constraint Satisfaction Problems and Database Theory. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) *Complexity of Constraints*. LNCS, vol. 5250, pp. 156–195. Springer, Heidelberg (2008)

Diversification and Intensification in Parallel SAT Solving

Long Guo¹, Youssef Hamadi^{2,3}, Said Jabbour⁴, and Lakhdar Sais¹

¹ Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens

[`{guo,sais}@cril.fr`](mailto:{guo,sais}@cril.fr)

² Microsoft Research
7 J J Thomson Avenue
Cambridge, United Kingdom

³ LIX École Polytechnique
F-91128 Palaiseau, France

[`youssefh@microsoft.com`](mailto:youssefh@microsoft.com)

⁴ INRIA-Microsoft Research Joint Centre
28 rue Jean Rostand

91893 Orsay Cedex, France

[`said.jabbour@inria.fr`](mailto:said.jabbour@inria.fr)

Abstract. In this paper, we explore the two well-known principles of diversification and intensification in portfolio-based parallel SAT solving. These dual concepts play an important role in several search algorithms including local search, and appear to be a key point in modern parallel SAT solvers. To study their trade-off, we define two roles for the computational units. Some of them classified as *Masters* perform an original search strategy, ensuring diversification. The remaining units, classified as *Slaves* are there to intensify their master's strategy. Several important questions have to be answered. The first one is what information should be given to a slave in order to intensify a given search effort? The second one is, how often, a subordinated unit has to receive such information? Finally, the question of finding the number of subordinated units and their connections with the search efforts has to be answered. Our results lead to an original intensification strategy which outperforms the best parallel SAT solver ManySAT, and solves some open SAT instances.

Keywords: Satisfiability, SAT and CSP, Search.

1 Introduction

In addition to the traditional hardware and software verification fields, SAT solvers are gaining popularity in new domains. For instance they are also used for general theorem proving and computational biology. This widespread adoption is the result of the efficiency gains made during the last decade [1]. Indeed, industrial instances with hundred of thousand of variables and millions of clauses are now solved within a few minutes. This impressive progress can be related to both the algorithmic improvements and to

the ability of SAT solvers to exploit the hidden structures¹ of such instances. However, new applications are always more challenging with instances of increasing size and complexity, while the gains traditionally given by low level algorithmic adjustments are now stalling. As a result, a large number of industrial instances from the last competitions remain challenging for all the available SAT solvers. Fortunately, this last challenge comes at a time where the generalization of multicore hardware gives parallel processing capabilities to standard PCs. While in general it is important for existing applications to exploit these new hardwares, for SAT solvers, this becomes crucial.

Many parallel SAT solvers have been previously proposed. Most of them are based on the divide-and-conquer principle (e.g. [2]). They generally divide the search space using the well known guiding-path concept [3]. The main problems behind these approaches rise in the difficulty to get workload balanced between the different processing units and in finding the best guiding path. Also, splitting the search tree using guiding paths leads to the exploration of unrelated parts of the search space and reduces the benefit of clauses sharing. Portfolio-based parallel SAT solving has been recently introduced [4]. It avoids the previous problem by letting several differentiated DPLL engines compete and cooperate to be the first to solve a given instance. Each solver works on the original formula, and search spaces are not split or decomposed anymore. To be efficient, the portfolio has to use diversified search engines with clauses sharing. The key point remains in finding such strategies while maintaining clauses exchange of higher quality. In ManySAT [4], the state-of-the-art portfolio-based parallel SAT solver, such diversification is obtained by a careful combination of different restarts policies, literals polarity assignment, and learning schemes. These differentiated search strategies enhanced with clause sharing aim to explore the search space with less possible redundancies. The first rank obtained by ManySAT on the parallel track of the 2008 SAT Race and 2009 SAT competition demonstrates that portfolio-based parallel approaches clearly outperform the divide-and-conquer based ones.

However, when clause sharing is added, diversification has to be restricted in order to maximize the impact of a foreign clause whose relevance is more important in a similar or related search effort. Despite the efficiency of ManySAT, the question of finding the best portfolio of diversified strategies while maintaining a high quality of exchange remains very challenging. Indeed, two orthogonal (respectively close) strategies might reduce (respectively increase) the impact of clause sharing. Therefore, a challenging question is to maintain a good and relevant "distance" between the parts of the search space explored by the different search units which is equivalent to the finding of a good diversification and intensification tradeoff. Indeed, intensification (respectively diversification) directs the search to the same (respectively different) parts of the search space. This question heavily depends on the problem instance. On hard ones it might be more convenient to direct the search towards building the same and common proof (intensification), whereas on easy ones diversifying it might be the way towards finding a short proof.

Taking this in mind, we propose to study the diversification/intensification tradeoff in a parallel SAT portfolio. We define two roles for the computational units. Some of them

¹ By structure, we understand the dependencies between variables, which can often appear through Boolean functions. One particular example being the well known notion of backdoors.

classified as *Masters* perform an original search strategy, ensuring diversification. The remaining ones, classified as *Slaves* are there to intensify their master's strategy. Doing so, several important questions have to be answered. The first one is what information should be given to a unit in order to intensify a given search effort? The second one is, how often, a subordinated unit has to receive such information? Finally, the question of finding the number of subordinated units along their connections with original search efforts has to be answered. In other words, we need to determine the best Masters/Slaves division and hierarchy i.e. topology.

In the following, Section two describes the internals of modern SAT solvers, and the architecture of a portfolio-based parallel SAT engine. Section three studies the best way to intensify a given search strategy. Section four, considers the different diversification/intensification tradeoffs in a portfolio. Section five, presents our experimental results. Finally, before the general conclusion, section six presents the related works.

2 Technical Background

In this section, we first introduce the most salient computational features of modern SAT solvers. Then, we describe a typical portfolio based parallel SAT solver.

2.1 Modern SAT Solvers

Modern SAT solvers [5,6], are based on classical DPLL search procedure [7] combined with (i) restart policies [8,9], (ii) activity-based variable selection heuristics (VSIDS-like) [5], and (iii) clause learning [10]. The interaction of these three components being performed through efficient data structures (e.g., Watched literals [5]).

Modern SAT solvers are especially efficient with "structured" SAT instances coming from industrial applications. On these problems, Gomes et al. [11] have identified a heavy tailed phenomenon, i.e., different variable orderings often lead to dramatic differences in solving time. This explains the introduction of restart policies in modern SAT solvers, which attempt to discover a good variable ordering. VSIDS and other variants of activity-based heuristics [12], on the other hand, were introduced to avoid thrashing and to focus the search: when dealing with instances of large size, these heuristics direct the search to the most constrained parts of the formula. VSIDS and restarts are two important and connected components since the first increase the activities of the variables involved in conflicts while the second allows the solver to reorder the decision stack according to these activities. Conflict Driven Clause Learning (CDCL) is the third component, leading to non-chronological backtracking. In CDCL a central data-structure is the *implication graph* [10], which records the partial assignment under construction made of the successive *decision literals* (chosen variable with either positive or negative *polarity*) with their propagations. Each time a conflict is encountered (say at level i) a *conflict clause* or nogood is learnt thanks to a bottom up traversal of the implication graph. Such a traversal can be seen as a resolution derivation starting from the two implications of the conflicting variable. The next resolvent is generated, from the previous one and another clause from the implication graph. Such linear resolution derivation stops when the current resolvent ($\alpha \vee a$) with α a sub-clause, contains only one literal a

from the current conflict level, called an *asserting literal*. The node in the graph labeled with $\neg a$ is called the *first Unique Implication Point* (first-UIP). This traversal or resolution process is also used to update the activity of related variables, allowing VSIDS to always select the most active variable as the new decision point. The learnt conflict clause $(\alpha \vee a)$, called *asserting clause*, is added to the learnt data base and the algorithm backtracks non chronologically to level $j < i$.

Modern SAT solvers can now handle propositional satisfiability problems with hundreds of thousands of variables or more. However, it is now recognized (see the recent SAT competitions) that the performances of the modern SAT solvers evolve in a marginal way. More precisely, on the industrial benchmarks category usually proposed to the annual SAT Races and/or SAT Competitions, many instances remain open (not solved by any solver within a reasonable amount of time). Consequently, new approaches are clearly needed to solve these challenging industrial problems.

2.2 ManySAT: A Parallel SAT Solver

ManySAT is a DPLL-engine which includes all the classical features like two-watched-literal, unit propagation, activity-based decision heuristics, lemma deletion strategies, and clause learning. In addition to the classical first-UIP scheme [13], it incorporates a new technique which extends the implication graph used during conflict-analysis to exploit the satisfied clauses of a formula [14]. Unlike other parallel SAT solvers, ManySAT does not implement a divide-and-conquer strategy based on some dynamic partitioning of the search space. On the contrary, it uses a portfolio philosophy which lets several sequential DPLLs compete and cooperate to be the first to solve the common instance. These DPLLs are differentiated in many ways. They use different and complementary restart strategies, VSIDS, polarity heuristics, and learning schemes. Additionally, all the DPLLs are exchanging learnt clauses up to some size limit.

As ManySAT finished first during the 2008 SAT Race and 2009 SAT Competition (parallel track - industrial category), we conducted our experimental comparison using this state-of-the-art parallel SAT solver.

3 Towards a Good Intensification Strategy

In this section, we first determine the relevant knowledge to be passed from a Master to a Slave in order to intensify the search. Secondly, we address the frequency of such directed intensification.

To this end, we consider a simple system with two computing units, respectively a Master (M) and a Slave (S) (see Figure 1). The role of the Master is to invoke the Slave for search intensification (dashed arrow in Figure 1). By intensification we mean that the slave would explore "differently" around the search space explored by the Master. Consequently, the clauses learnt by the Master and the Slave are relevant to each other and shared in both direction (plain line in Figure 1).

To explore differently around a given search effort, several kind of knowledge can be considered. Suppose that the Master is currently at a given state $S_M = (\mathcal{F}, \mathcal{D}_M, \Gamma_M)$, where \mathcal{F} is the original SAT instance, \mathcal{D}_M the set of decision literals, and Γ_M the set

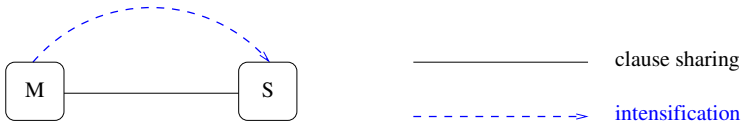


Fig. 1. Intensification topology

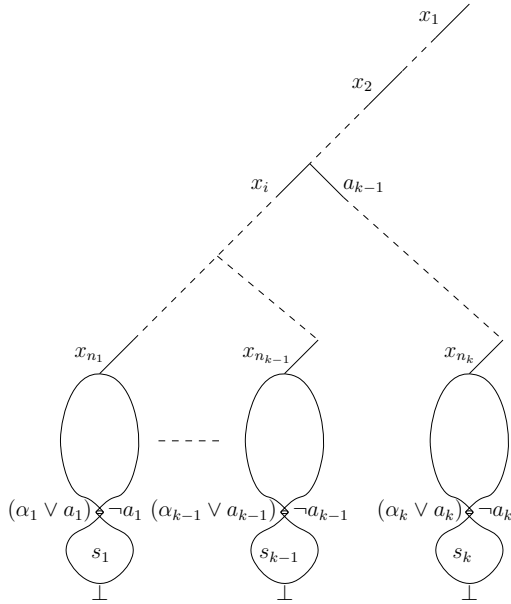


Fig. 2. A partial view of the Master search tree : conflicts branches and implication graphs

of learnt clauses (learnt database). In the following, from a given state S_M , we derive three different knowledge characterizing the Master search effort.

We use Figure 2 to illustrate such knowledge. It represents a current state S_M corresponding to the branch leading to the last conflict k . The decisions made in the last branch are x_1, x_2, \dots, x_{n_k} . The boxes give a partial view of the implication graph obtained on the last k conflicts derived after the assignment of the last decisions $x_{n_k}, x_{n_{k-1}}, \dots,$ and x_{n_1} . The learnt clauses are respectively $(\alpha_k \vee a_k), (\alpha_{k-1} \vee a_{k-1}), \dots,$ and $(\alpha_1 \vee a_1)$ where $a_k, a_{k-1}, \dots,$ and a_1 are the asserting literals corresponding to the first-UIP $\neg a_k, \neg a_{k-1}, \dots,$ and $\neg a_1$.

Decision list. The first kind of knowledge characterizing the Master search effort uses the current set of decisions \mathcal{D}_M (in short *decision list*). Using such decisions, the Slave can build the whole or a subset of the current partial assignment of the Master depending if all the asserting clauses generated by M on the current branch are passed to S. Since the activity of the variables are not passed to the Slave, it shall explore the same area in a different way.

Asserting set. The second one, uses the sequence $A_M = \langle a_k, a_{k-1}, \dots, a_1 \rangle$ (in short *asserting set*) of the Master asserting literals associated to the k clauses learnt before the current state S_M . The sequence is ordered from the latest to the oldest conflict. By branching on the ordered sequence A_M using the same polarity, the Slave is able to construct a partial assignment involving the most recent asserting literals learnt from the Master unit. Let us recall that an asserting literal a_i is part of the Master learnt clause $(\alpha \vee a_i)$. As the Slave branches on a_i , future conflicts analysis involving a_i , might lead to learnt clauses containing $\neg a_i$. More generally, invoking the Slave using A_M pushes it to learn more relevant clauses, connected by resolution (contains complementary literals) to the most recent clauses learned by M . This is clearly an intensification process, as the clauses learnt by S involve the most important literals of M , and lead in some way to a more constructive resolution proof thanks to the complementary shared literals between M 's learnt clauses, and the future clauses that will be learnt by S .

Conflict sets. The last one, uses the sequence of ordered sets $C_M = \langle s_k, s_{k-1}, \dots, s_1 \rangle$ of literals collected during the Master conflict analysis (in short *conflict sets*). The set s_k represents the set of literals collected during the last conflict analysis. More precisely, the literals in s_k correspond to the nodes of the implication graph located between the conflict node and the first-UIP node $\neg a_k$ (see Figure 2). Moreover, the set s_k includes a literal of the conflicting variable and the literal labeling the first-UIP node $\neg a_k$. It can be defined as $s_k = \langle y_{k_1}, y_{k_2}, \dots, y_{k_m} \rangle$, where y_{k_1} corresponds to the literal of the first-UIP node $\neg a_k$ and y_{k_m} to the literal of the conflict variable as it appears in the current partial assignment. The aim of considering this sequence of sets is to intensify the search by directing S around the same conflicts. Let us note that the activity of the variables appearing in the conflict sets are those updated during conflict analysis. One can use the most active variables of the Masters to direct the search of the Slaves. However, exploiting such kind of knowledge leads to redundant search between the Masters and Slaves i.e. the Masters and Slaves tends to reproduce the same search.

We can first remark that, the sequence A_M and C_M might contain redundant literals (the same literal occurs several times). As the Slave S assign such literals according to the defined ordering, S chooses the next unassigned literal in the ordering. For the first one, and as mentioned above, the Master invoke the Slave using the decision list D_M together with the set of asserting clauses learnt on the current branch in order to build the same partial assignment.

To compare the relevance of the previously defined intensification strategies, we conducted the following experiments on the whole set of instances (292 instances) from the industrial category of the 2009 SAT Competition. We use ManySAT with two computing units (see figure 1) sharing clauses of size less or equal to 8. The Master M invokes the Slave S at each restart and transmits at the same time the intensification knowledge. For the Master M we used a rapid restart strategy. It is widely admitted that rapid restarts lead to better learning [15] or to learnt clauses of small width [16]. Additionally, rapid restarts provide frequent intensification of the Slave leading to a tight synchronization of the search efforts.

Let us note that, the Slave do not implement any restart strategy. It restarts when invoked by the Master. For the Master, we use in this experiments the rapid and dynamic restart policy introduced in [4].

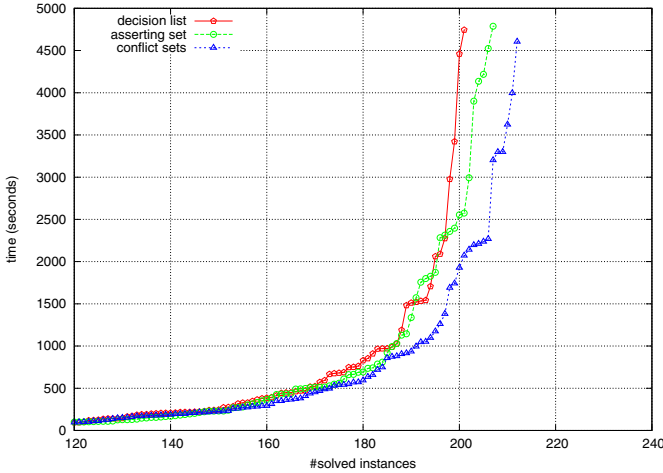


Fig. 3. Three intensification strategies

The Figure 3 shows the experimental comparison using the above three intensification strategies (*decision list*, *asserting set*, and *conflict sets*). It presents the cumulated time results i.e. the number of instances (x-axis) solved under a given amount of time in seconds (y-axis). As we can observe, directing the search using *conflict sets* gives the best results. The number of solved instances using the *decision list*, *asserting set* and *conflict sets* are 201, 207 and 212 respectively. In the rest of this paper, we use *conflict sets* as the intensification strategy.

4 Towards a Good Search Tradeoff

This section explores the diversification and intensification tradeoff. We are using the ManySAT architecture which is represented by a clique of four computational units interacting through clause sharing [4] up to size 8. As ManySAT finished first during the 2008 SAT Race and 2009 SAT Competition (parallel track - industrial category), we are testing our intensification technique against a state-of-the-art solver. These units represent a fully diversified set of strategies. In order to add some intensification, we propose to extend this architecture and to partition the units between Masters and Slaves. If we allow a Slave to intensify its own search effort through another Slave, we have a total of seven possible configurations. They are presented in Figure 4. In this Figure, dotted lines represents the Master/Slave relationships. Note that when a unit has to provide intensification directives to several Slaves, it alternates its guidance between them, i.e., round-table. Moreover, when a configuration contains chain(s) of Slaves, (see (d), (f), and (g) in Figure 4), the intensification of a Slave of level i is triggered by the Slave of level $i - 1$.

These configurations represent all the possible diversification and intensification tradeoffs which can be implemented on top of the ManySAT architecture. We recall

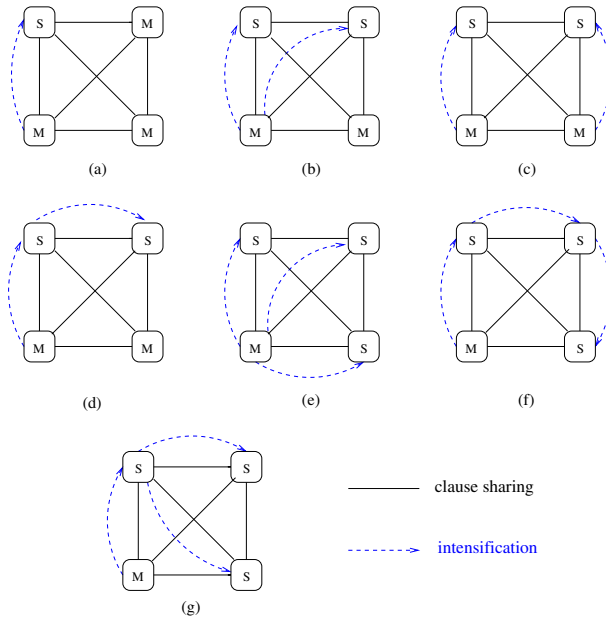


Fig. 4. Diversification/Intensification topologies

that ManySAT exploit diversified search strategies on each core [4]. In ManySAT, the different cores (or processing units) are ordered according to their overall performance from the best (core 0) to the least best (core 3). The performance of the different cores are taken from the results obtained by ManySAT during the last SAT 2009 competition and corresponds to the number of instances solved by each core. In the different topologies of Figure 4, the core 0, core 1, core 2 and core 3 corresponds to the processing unit at the bottom left, bottom right, top right and top left boxes respectively. Naturally, in our experiments, we allocate in priority the best strategies of ManySAT to Masters and the least performant ones to Slaves. This rational choice avoids to consider all the possible symmetric topologies that can be obtained by simple rotations.

The following section explores their respective performances and compare them to the original ManySAT solver.

5 Experiments

Our tests were done on Intel Xeon quadcore machines with 32GB of RAM running at 2.66 Ghz. For each instance, we used a timeout of 4 hours of CPU time which corresponds to a 1 hour timeout per computational unit (core). Our Master/Slave roles and their different configurations were implemented on top of the original ManySAT. This solver was also used as a baseline for comparison. We used the *conflict sets* intensification strategy.

We used the 292 industrial instances of the 2009 SAT competition to compare our different algorithms.

Table 1. 2009 SAT Competition, Industrials: overall results

Method	# SAT	# UNSAT	Total	Tot. time (sc.)	Avg. time
ManySAT	87	125	212	329378	1128
Topo. (a)	86 (7)	133 (49)	219 (56)	311590	1067
Topo. (b)	84 (28)	130 (73)	214 (101)	324800	1112
Topo. (c)	89 (23)	132 (74)	221 (97)	307345	1052
Topo. (d)	87 (25)	132 (67)	219 (92)	315537	1080
Topo. (e)	86 (45)	131 (109)	217 (154)	323208	1106
Topo. (f)	82 (44)	128 (102)	210 (146)	339677	1163
Topo. (g)	80 (45)	127 (107)	207 (152)	343800	1177

The Table 1 summarizes our results. The first column presents the method, i.e., the original ManySAT (first line) or ManySAT extended with one of our seven diversification/intensification topology (see Figure 4). In the second column, the first number represents the overall number of SAT instances solved by the associated method, the second number (in parenthesis) gives the number of instances found SAT by a Slave. The third column gives similar information for UNSAT problems. The column four, gives the overall number of instances solved, again the parenthesis gives the number solved by one of the Slaves. To alleviate the effects of unpredictable threads scheduling, each instance was solved three times and we take the average as the time needed for solving a given instance. The average is calculated using the 1 hour timeout when an instance is not solved at a given run. Finally, the last two columns give respectively, the total time (cumulated), and the average time in seconds calculated over the overall set of 292 instances.

This Table shows that the vast majority of our topology-based extensions are superior to the original ManySAT. This algorithm solves 212 problems whereas the best topology (c) solves 221. Remarkably, all the topologies are able to solve more UNSAT problems than ManySAT. This unsurprisingly shows that adding intensification, is more beneficial on this last category of problems. Indeed, our intensification strategy increases the relevance of the learnt clauses exchanged between masters and slaves, since unsatisfiable instances are mainly solved by resolution, improving the quality of the learnt clauses increases the performances on UNSAT problems.

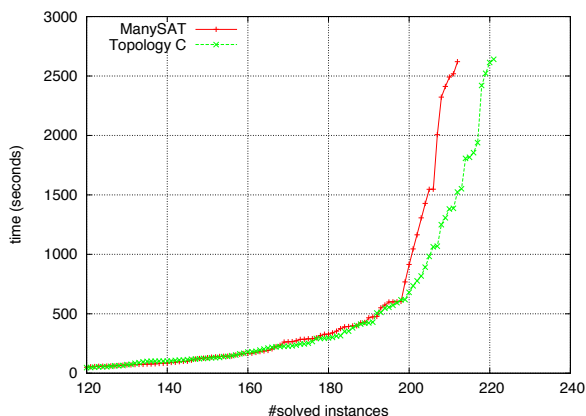
When we compare the results achieved by our different topologies. It seems that balancing the tradeoff between 2 Slaves and 2 Masters works better (topo. b, c, and d). Among them, balancing the slaves to the masters gives the most efficient results i.e., topology c.

The Table 2 highlights the results achieved by our best topology (c) against ManySAT on three complete families of problems. We can see that our best topology outperforms ManySAT on all these problems. Let us mention that we have not found families where ManySAT dominates our best topology (c). Even more importantly, our algorithm allowed the resolution of two open instances (9dlx_vliw_at_b_iq8, and 9dlx_vliw_at_b_iq9), proved UNSAT for the first time.

The Figure 5 presents cumulated time results for ManySAT and for our best topology on the whole set of problems. On small time limit (less than 10 minutes), the algorithms have the same behavior. On the other hand, when more time is allowed, the new technique exhibits an important improvement, and solves 9 more instances.

Table 2. 2009 SAT Competition, Industrials: time (s) results on three families

Instance	Status	ManySAT	Topology (c)
9dlx_vliw_at_b_iq1	UNSAT	87.3	10.6
9dlx_vliw_at_b_iq2	UNSAT	226.3	27.1
9dlx_vliw_at_b_iq3	UNSAT	602.8	103.2
9dlx_vliw_at_b_iq4	UNSAT	1132	163.5
9dlx_vliw_at_b_iq5	UNSAT	2428	313.1
9dlx_vliw_at_b_iq6	UNSAT	–	735.6
9dlx_vliw_at_b_iq7	UNSAT	–	991
9dlx_vliw_at_b_iq8	UNSAT	–	1822.7
9dlx_vliw_at_b_iq9	UNSAT	–	2670.1
velev-pipe-sat-1.0-b10	SAT	4.4	3.6
velev-engi-uns-1.0-4nd	UNSAT	5	4.9
velev-live-uns-2.0-ebuf	UNSAT	6.7	6.8
velev-pipe-sat-1.0-b7	SAT	48.3	6.2
velev-pipe-o-uns-1.1-6	UNSAT	65.2	30.8
velev-pipe-o-uns-1.0-7	UNSAT	149.9	118.2
velev-pipe-uns-1.0-8	UNSAT	274.5	82.7
velev-vliw-uns-4.0-9C1	UNSAT	297.2	235.4
velev-vliw-uns-4.0-9-i1	UNSAT	–	1311.6
goldb-heqc-term1mul	UNSAT	23.8	4.3
goldb-heqc-i10mul	UNSAT	36.3	23.5
goldb-heqc-alu4mul	UNSAT	49.9	40.9
goldb-heqc-dalumul	UNSAT	384.1	33.6
goldb-heqc-frg1mul	UNSAT	2606	83.1
goldb-heqc-x1mul	UNSAT	–	246.9

**Fig. 5.** 2009 SAT Competition, Industrials: cumulated time

Finally, it is important to note that in the last SAT 2009 competition no sequential or parallel SAT solver has been able to reach such number of solved instances. In SAT 2009 competition, all the solvers are allowed a time limit of about 3 hours (10 000 seconds) for a given instance. The tests were done on a Intel Xeon machines with 2GB of RAM and 3.2 Ghz. The Virtual Best Solver (VBS) solved 229 instances (91 SAT and 138 UNSAT). VBS is a theoretical construction which returns the best answer provided by one of the submitted solver. An instance is solved by VBS if it is solved by at least one of the submitted solvers. Another way to look at it is to consider this VBS as a

solver which would run all other solvers in parallel, bringing together all the solvers strengths. This VBS is essentially the same notion as State Of The Art (SOTA) solver defined in [17]. From the description above, we can measure that the performance of our proposed approach is very close to those of VBS.

6 Related Works

We present here the most noticeable approaches related to parallel SAT solving.

In [18] a parallelization scheme for a class of SAT solvers based on the DPLL procedure is presented. The scheme uses a dynamic load-balancing mechanism based on work-stealing techniques to deal with the irregularity of SAT problems. PSatz is the parallel version of the well known Satz solver. Gradsat [19] is based on zChaff. It uses a master-slave model and the notion of guiding-paths to split the search space and to dynamically spread the load between clients. Learned clauses are exchanged between all clients if they are smaller than a predefined limit on the number of literals. A client incorporates a foreign clause when it backtracks to level 1 (top-level).

[20] uses an architecture similar to Gradsat. However, a client incorporates a foreign clause if it is not subsumed by the current guiding-path constraints. Practically, clause sharing is implemented by *mobile-agents*. This approach is supposed to scale well on computational grids.

In [21], the input formula is dynamically divided into disjoint subformulas. Each subformula is solved by a sequential SAT-solver running on a particular processor. The algorithm uses optimized data structures to modify Boolean formulas. Additionally workload balancing algorithms are used to achieve a uniform distribution of workload among the processors.

MiraXT [2], is designed for shared memory multiprocessors systems. It uses a divide and conquer approach where threads share a unique clause database which represents the original and the learnt clauses. When a new clause is learnt by a thread, it uses a lock to safely update the common database. Read access can be done in parallel.

PMSat uses a master-slave scenario to implement a classical divide-and-conquer search [22]. The user of the solver can select among several partitioning heuristics. Learnt clauses are shared between workers, and can also be used to stop efforts related to search spaces that have been proven irrelevant. PMSat runs on networks of computer through an MPI implementation.

[23] uses a standard divide-and-conquer approach based on guiding-paths. However, it exploits the knowledge on these paths to improve clause sharing. Indeed, clauses can be large with respect to some static limit, but when considered with the knowledge of the guiding path of a particular thread, a clause can become small and therefore highly relevant. This allows pMiniSat to extend the sharing of clauses since a large clause can become small in another search context.

In [24] a SAT Solver *c-sat*, a parallelization of MiniSat using MPI is presented. It employs a layered master-worker architecture, where the masters handle lemma exchange, deletion of redundant lemmas and the dynamic partitioning of search trees, while the workers do search using different decision heuristics and random number seeds.

In [25] a new switching criterion based on the evenness or unevenness of the distribution of variable weights is presented. The proposed hybrid local search algorithm

combines intensification and diversification by switching between two different heuristics using this criterion.

Other portfolio-based solvers have been proposed in the sequential context, such as Satzilla [26] or cpHydra [27], they mainly based on running several solvers on a set of training instances in order to determine the most appropriate solver to solve a given instance. They clearly differ from parallel portfolio based solvers, where all the solvers of the portfolio are run in parallel and clauses are shared between them.

7 Conclusion

We have explored the two well-known principles of diversification and intensification in portfolio-based parallel SAT solving. These dual concepts play an important role in several search algorithms including local search, and appear to be a key point in modern parallel SAT solvers. To study their tradeoff, we defined two roles for the computational units. Some of them classified as *Masters* perform an original search strategy, ensuring diversification. The remaining units, classified as *Slaves* are there to intensify their master's strategy.

Several important questions have been addressed. The first one is what information should be given to a slave in order to intensify a given search effort? It appeared that passing the set of literals found during previous conflict analysis gives the best results. This strategy aims at directing the slave towards conflicts highly related to the master's conflicts, allowing masters and slaves to share highly relevant clauses.

The second one is, how often, a subordinated unit has to receive such information? We have decided to exploit the restart policy of a master to refresh the information given to its slave(s). As shown in other works, rapid restarts lead to better learning [15] or to learnt clauses of small width [16]. Therefore, a rapid restarts strategy on the master node reinforces the interests of the clauses shared with its slaves. In our context it allows frequent intensification of a Slave leading to a tight synchronization of the search efforts.

Finally, the question of finding the number of subordinated units along their connections with the search efforts had to be answered. Our tests have shown that balancing the set of nodes between Masters and Slaves roles, and balancing the slaves to the masters gives the best results. In particular, our best topology solves 9 more industrial instances than the actual best solver, ManySAT. The results have also demonstrated the relative performance of the intensification strategy on UNSAT problems. Remarkably, our new strategy was able to close the 9dlx_vliw_at_b.iq* family by finding the proofs of unsatisfiability for two open instances.

As future work, we would like to dynamically adapt the topology and roles in a portfolio based on the perceived hardness of a given instance. This should benefit to hard UNSAT proofs where several units could be used for intensification, and at the same time, could preserve performances on difficult SAT problems where intensification is less needed. A second interesting path for future research concerns the integration of control based clause sharing [28] in this context. The third issue is to address scalability, one of the most important challenge in parallel SAT solving. The framework proposed in this paper is better suited to achieve this goal. Indeed, as the intensification leads to clause sharing of better quality, we can allow such exchange between a

Master and its Slave only. This will reduce the number of shared clauses while maintaining the overall performance. Other measures of clause-quality need to be defined in order to reduce the global number of exchanged clauses. Finally, we plan to extend our proposed framework for solving other problems around SAT including constraint satisfaction problems.

References

1. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
2. Lewis, M.D.T., Schubert, T., Becker, B.: Multithreaded sat solving. In: ASP-DAC, pp. 926–931 (2007)
3. Zhang, H., Bonacina, M.P., Hsiang, J.: Psato: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation* 21, 543–560 (1996)
4. Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a parallel SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation - JSAT* 6, 245–262 (2009)
5. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC 2001), pp. 530–535 (2001)
6. Eén, N., Sörensson, N.: An extensible sat-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
7. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. *Communications of the ACM* 5(7), 394–397 (1962)
8. Gomes, C.P., Selman, B., Kautz, H.A.: Boosting combinatorial search through randomization. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI 1998), Madison, Wisconsin, pp. 431–437 (1998)
9. Kautz, H.A., Horvitz, E., Ruan, Y., Gomes, C.P., Selman, B.: Dynamic restart policies. In: AAAI/IAAI, pp. 674–681 (2002)
10. Marques-Silva, J.P., Sakallah, K.A.: GRASP - A New Search Algorithm for Satisfiability. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp. 220–227 (November 1996)
11. Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reasoning* 24(1/2), 67–100 (2000)
12. Brisoux, L., Grégoire, É., Sais, L.: Improving backtrack search for SAT by means of redundancy. In: Raś, Z.W., Skowron, A. (eds.) ISMIS 1999. LNCS, vol. 1609, pp. 301–309. Springer, Heidelberg (1999)
13. Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in boolean satisfiability solver. In: ICCAD, pp. 279–285 (2001)
14. Audemard, G., Bordeaux, L., Hamadi, Y., Jabbour, S., Sais, L.: A generalized framework for conflict analysis. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 21–27. Springer, Heidelberg (2008)
15. Biere, A.: Adaptive restart strategies for conflict driven sat solvers. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 28–33. Springer, Heidelberg (2008)
16. Pipatsrisawat, K., Darwiche, A.: Width-based restart policies for clause-learning satisfiability solvers. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 341–355. Springer, Heidelberg (2009)
17. Sutcliffe, G., Suttner, C.B.: Evaluating general purpose automated theorem proving systems. *Artificial Intelligence* 131(1-2), 39–54 (2001)

18. Jurkowiak, B., Li, C.M., Utard, G.: A parallelization scheme based on work stealing for a class of sat solvers. *Journal of Automated Reasoning* 34(1), 73–101 (2005)
19. Chrabakh, W., Wolski, R.: GrADSAT: A parallel sat solver for the grid. Technical report, UCSB Computer Science Technical Report Number 2003-05 (2003)
20. Blochinger, W., Sinz, C., Küchlin, W.: Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Computing* 29(7), 969–994 (2003)
21. Böhm, M., Speckenmeyer, E.: A fast parallel sat-solver - efficient workload balancing. *Annals of Mathematics and Artificial Intelligence* 17(3-4), 381–400 (1996)
22. Gil, L., Flores, P., Silveira, L.M.: PMSat: a parallel version of minisat. *Journal on Satisfiability, Boolean Modeling and Computation* 6, 71–98 (2008)
23. Chu, G., Stuckey, P.J.: Pminisat: a parallelization of minisat 2.0. Technical report, Sat-race 2008, solver description (2008)
24. Ohmura, K., Ueda, K.: c-sat: A parallel sat solver for clusters. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 524–537. Springer, Heidelberg (2009)
25. Wei, W., Li, C.M., Zhang, H.: A switching criterion for intensification and diversification in local search for sat. *Journal on Satisfiability, Boolean Modeling and Computation - JSAT* 4(2-4), 219–237 (2008)
26. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research (JAIR)* 32, 565–606 (2008)
27. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science, AICS 2008* (2008)
28. Hamadi, Y., Jabbour, S., Sais, L.: Control-based clause sharing in parallel sat solving. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 499–504 (2009)

A Systematic Approach to MDD-Based Constraint Programming

Samid Hoda, Willem-Jan van Hoeve, and J.N. Hooker

Tepper School of Business, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A.

{shoda,vanhoeve}@andrew.cmu.edu, john@hooker.tepper.cmu.edu

Abstract. Fixed-width MDDs were introduced recently as a more refined alternative for the domain store to represent partial solutions to CSPs. In this work, we present a systematic approach to MDD-based constraint programming. First, we introduce a generic scheme for constraint propagation in MDDs. We show that all previously known propagation algorithms for MDDs can be expressed using this scheme. Moreover, we use the scheme to produce algorithms for a number of other constraints, including `Among`, `Element`, and unary resource constraints. Finally, we discuss an implementation of our MDD-based CP solver, and provide experimental evidence of the benefits of MDD-based constraint programming.

1 Introduction

The domain store is a fundamental tool for constraint programming (CP), because it propagates the results of individual constraint processing. It allows the reduced domains obtained for one constraint to be passed to the next constraint for further filtering. A weakness of the domain store, however, is that it transmits a limited amount of information. It accounts for no interaction among the variables, because any solution in the Cartesian product of the current domains is consistent with it. This restricts the ability of the domain store to pool the results of processing individual constraints and provide a global view of the problem.

To address this shortcoming, Andersen, Hadzic, Hooker, and Tiedemann [1] proposed replacing the domain store with a richer data structure, namely a multivalued decision diagram (MDD). In their approach, domain filtering algorithms are replaced or augmented by algorithms that refine and update the MDD to reflect each constraint. It was found that MDD-based propagation can lead to substantial speedups in the solution of multiple `AllDifferent` constraints. The idea was extended to equality constraints by Hadzic et al. [4]. A unified node-splitting scheme for refining the MDD was proposed by Hadzic et al. [3] and applied to certain configuration problems. For this reason, we will mainly focus on filtering algorithms in this work.

MDDs have been applied before in CP. For example, in [6] and [2] MDDs are applied to perform inferences (domain filtering) based on individual constraints.

In [5], this is taken one step further, by passing structural information from one constraint to the next. The key difference is that in these approaches, an MDD is built and maintained for each individual constraint, whereas in MDD-based constraint programming, the MDD is the information that is passed from one constraint to the next. In other words, multiple constraints process the same MDD, instead of each constraint processing its individual MDD.

The contributions of this work are threefold. First, we introduce a systematic scheme for constraint propagation in MDDs, and we show that all previously proposed filtering algorithms for MDD-based CP can be viewed as instantiations of our scheme. Second, we apply our scheme to introduce new filtering algorithms for other constraints; we present such algorithms for the **Among**, **Element**, and unary resource constraints as an illustration of the versatility of the approach. Third, we present computational results for the first pure MDD-based CP solver, showing that *i*) this approach can scale up to realistic problem sizes, and *ii*) enormous savings (in terms of time as well as search tree size) can be realized when compared to solvers relying on the traditional domain store.

The remainder of the paper is organized as follows. In Section 2 we provide the necessary background on MDDs and MDD-based constraint programming. In Section 3 we present and discuss a systematic scheme for constraint propagation in MDDs. We apply our scheme to a variety of constraints in Section 4. In Section 5 we provide a brief description of our MDD-based constraint programming system. Experimental results using this system are reported in Section 6. Finally, we conclude in Section 7.

2 MDDs and MDD-Based Constraint Solving

In this work, an *ordered Multivalued Decision Diagram (MDD)* is a directed acyclic graph whose nodes are partitioned into n (possibly empty) subsets or *layers* L_1, \dots, L_{n+1} , where the layers L_1, \dots, L_n correspond respectively to variables x_1, \dots, x_n . L_1 contains a single *top* node \mathbf{T} , and L_{n+1} contains two *bottom* nodes $\mathbf{0}$ and $\mathbf{1}$. The *width* of the MDD is the maximum number of nodes in a layer, or $\max_{i=1}^n \{|L_i|\}$. In MDD-based CP, the MDDs typically have a given fixed maximum width.

All edges of the MDD are directed from an upper to a lower layer; that is, from a node in some L_i to a node in some L_j with $i < j$. For our purposes it is convenient to assume (without loss of generality) that each edge connects two adjacent layers. Let $L(s)$ denote the layer of the node s , and $\text{var}(s)$ the variable associated with $L(s)$. Each edge out of layer i is labeled with an element of the domain $D(x_i)$ of x_i . The set $E(s, t)$ of edges from node s to node t may contain multiple edges, and we denote each with its label. Let $E^{in}(s)$ denote the set of edges coming into s , and $E^{out}(s)$ the set of edges leaving s . For an edge e , $\text{tail}(e)$ is the tail of e and $\text{head}(e)$ the head.

An edge with label v leaving a node in layer i represents an assignment $x_i = v$. Each path in the MDD from \mathbf{T} to $\mathbf{0}$ or $\mathbf{1}$ can be denoted by the edge labels v_1, \dots, v_n on the path and is identified with the assignment (x_1, \dots, x_n)

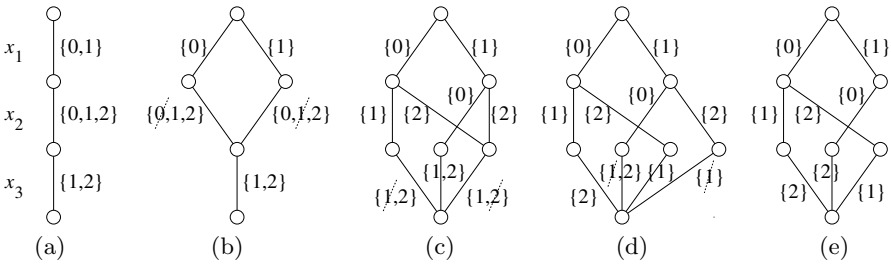


Fig. 1. Refining and filtering an MDD of width one (a) for $x_1 \neq x_2$ (b), $x_2 \neq x_3$ (c), and $x_1 \neq x_3$ (d), yielding the MDD in (e). Dashed lines mark filtered values.

$= (v_1, \dots, v_n)$. For our purposes, it is convenient to generate only the portion of an MDD that contains paths from \mathbf{T} to $\mathbf{1}$. A path v_1, \dots, v_n is *feasible* for a given constraint C if setting $(x_1, \dots, x_n) = (v_1, \dots, v_n)$ satisfies C . Constraint C is feasible on an MDD if the MDD contains a feasible path for C .

A constraint C is called *MDD consistent* on a given MDD if every edge of the MDD lies on some feasible path. Thus MDD consistency is achieved when all redundant edges (i.e., edges on no feasible path) have been removed. Domain consistency for C is equivalent to MDD consistency on an MDD of width one that represents the variable domains. That is, it is equivalent to MDD consistency on an MDD in which each layer L_i contains a single node s_i , and $E(s_i, s_{i+1}) = D(x_i)$ for $i = 1, \dots, n$.

Typically, MDD-based constraint programming starts with simple MDD (of width one) that permits all solutions represented by the Cartesian product of the domains. This MDD is then *refined* each time a constraint is processed. Refinement is accomplished by adding some nodes and edges to the MDD so as to exclude solutions that violate the constraint. Example 1 below gives an illustration of this (see also Figure 1).

The basic operation of refinement is *node-splitting*, in which the edges entering a given node are partitioned into equivalence classes, and ideally the node is split into one copy for each equivalence class. The set of outgoing edges for each copy is the same as the set of outgoing edges of the original node. We note that determining the equivalence classes may be costly to compute in practice, in which case an approximation of equivalence is used. We take care that the width of the MDD (maximum number of nodes in a layer) remains within a fixed bound. When splitting a node we merge equivalence classes when necessary in order to respect this restriction. The resulting MDD is a relaxation in the sense that it may fail to exclude all assignments that violate the constraint, but it is a much stronger relaxation than a domain store. A principled approach to node refinement in MDDs is introduced by Hadzic et al. [3].

We also update the MDD by deleting infeasible edges, an operation that generalizes conventional domain filtering. We will refer to this operation as *MDD filtering*. This can lead to further reduction of the MDD, if after the removal of the edge some other edges no longer have a path to $\mathbf{1}$ or can no longer be

reached by a path from the root. An MDD-based constraint solver is based on *propagation* and *search* just as traditional CSP solvers, but the domain filtering process at each node of the search tree is replaced (or supplemented) by an MDD refinement and filtering process.

Example 1. Consider a CSP with variables $x_1 \in \{0, 1\}$, $x_2 \in \{0, 1, 2\}$, and $x_3 \in \{1, 2\}$, and constraints $x_1 \neq x_2$, $x_2 \neq x_3$, and $x_1 \neq x_3$. All domain values are domain consistent (even if we were to apply the **AllDifferent** propagator on the conjunction of the constraints), and the domain store defines the relaxation $\{0, 1\} \times \{0, 1, 2\} \times \{1, 2\}$, which includes infeasible solutions such as $(1, 1, 1)$.

The MDD-based approach starts with the MDD of width one in Figure 1(a), in which multiple arcs are represented by a set of corresponding domain values for clarity. We refine and filter each constraint separately. Starting with $x_1 \neq x_2$, we refine the MDD by splitting the node at layer 2, resulting in Figure 1(b). This allows us to filter two domain values, based on $x_1 \neq x_2$, as indicated in the figure. In Figure 1(c) and (d) we refine and filter the MDD for the constraints $x_2 \neq x_3$ and $x_1 \neq x_3$ respectively, until we reach the MDD in Figure 1(e). This MDD represents all three solutions to the problem, and provides a much tighter relaxation than the domain store.

3 A Systematic Scheme for MDD Propagation

In the literature, MDD propagation algorithms (in the sense of Section 2) have been proposed for the following three constraint types: (one-sided) inequality constraints 1, **AllDifferent** 1, and equality constraints 3. The reasoning used for designing propagation algorithms for each of these constraints seemed to be ad-hoc. In this section we will present and analyze a systematic scheme for designing MDD propagation algorithms. In the following section we use this procedure to express the existing MDD propagation algorithms and introduce new algorithms for the **Among**, **Element**, and unary resource constraints.

3.1 The General Scheme

Our scheme is based on the idea of ‘local information’ $I(s)$ stored for each constraint at each node s . We will show that all MDD propagation schemes so far introduced, and others as well, can be viewed as based on local information. The precise nature of the local information depends on the propagation scheme, which is characterized in part by what kind of local information is required to apply it.

More precisely, the decision as to whether to delete an edge in $E(s, t)$ is based solely on $I(s)$ and $I(t)$ —that is, on local information stored at either end of the edge. Furthermore, the local information can be accumulated by a single top-down pass and a single bottom-up pass through the MDD.

It is convenient to regard $I(s)$ as a pair $(I^\downarrow(s), I^\uparrow(s))$ consisting of the information $I^\downarrow(s)$ accumulated during the top-down pass, and the information $I^\uparrow(s)$ accumulated during the bottom-up pass. $I^\downarrow(s)$ and $I^\uparrow(s)$ can take several forms,

such as a set of domain values or parameters related to the constraint, or a tuple of such sets. What is common to all the schemes is that $I^\downarrow(s)$ is computed solely on the basis of local information at the opposite end of edges coming into s , and $I^\uparrow(s)$ on the basis of local information at the opposite end of edges leaving s .

Formally, we introduce an operation \otimes that processes information when traversing an edge during a top-down or bottom-up pass. When traversing an edge $e \in E(s, t)$ during the top-down pass, the information $I^\downarrow(s)$ at node s is combined with edge e to obtain updated information $I^\downarrow(s) \otimes e$. We view this updated information as an object I having the same form as $I^\downarrow(s)$. When several edges enter node t , we use an operation \oplus to combine the information obtained by traversing the incoming edges. The top-down information at t is therefore

$$I^\downarrow(t) = \bigoplus_{e \in E^{in}(t)} I^\downarrow(\text{tail}(e)) \otimes e$$

Similarly, the bottom-up information at s is

$$I^\uparrow(s) = \bigoplus_{e \in E^{out}(s)} I^\uparrow(\text{head}(e)) \otimes e$$

Several examples of this scheme appear in the following sections.

Since a top-down (bottom-up) pass of the MDD visits each edge exactly once, the passes themselves involve an amount of work that is linear in the size of the MDD (modulo the work required to compute \oplus and \otimes at each node).

The operators \otimes and \oplus can be implemented as high-level macros that are instantiated differently for each constraint type. Our MDD-based CP solver follows this idea very closely.

3.2 MDD Consistency

The scheme above can sometimes achieve MDD consistency in polynomial time. In particular, if it can determine in polytime whether any particular assignment $x_j = v$ is consistent with the MDD, then it achieves MDD consistency in polytime due to the following theorem.

Theorem 1. *Suppose that the feasibility of $x_j = v$ for a given constraint C on a given MDD M can be determined in $\mathcal{O}(f(M))$ time and space for any variable x_j in C and any $v \in D(x_j)$. Then we can achieve MDD consistency for C in time and space at most $\mathcal{O}(\text{poly}(M)f(M))$.*

The proof is a straightforward shaving argument. For each edge e of M we consider the MDD M_e that consists of all the **T**-to-**1** paths in M containing e . Then e can be removed from M if and only if $x_j = e$ is inconsistent with C and M_e , where $j = L(\text{tail}(e))$. This can be determined in time and space at most $\mathcal{O}(f(M_e)) \leq \mathcal{O}(f(M))$. By repeating this operation $\text{poly}(M)$ times (i.e., on each edge of M) we obtain the theorem.

To establish MDD consistency, the goal is to efficiently compute information that is strong enough to apply Theorem 1. In the sequel, we will see that this can

be done for inequality constraints and **Among** constraints in polynomial time, and in pseudo-polynomial time for two-sided inequality constraints. Furthermore, we have the following result.

Corollary 1. *For binary constraints that are given in extension, our scheme can be applied to achieve MDD consistency in polynomial time (in the size of the constraint and the MDD).*

To see how this is accomplished, suppose C is a binary constraint containing variables x_i, x_j for $i < j$ (the argument for $j < i$ is similar). We let $I^\downarrow(s)$ contain all the values assigned to x_i in some path from \mathbf{T} to s , and $I^\uparrow(t)$ contain all the values assigned to x_j in some path from s to $\mathbf{1}$. Then we can delete edge $e \in E(s, t)$ from M if and only if (a) $L(s) = j$ and $(x_i, x_j) = (v, e)$ satisfies C for no $v \in I^\downarrow(s)$, or (b) $L(s) = i$ and $(x_i, x_j) = (e, v)$ satisfies C for no $v \in I^\uparrow(t)$. Because $|I^\downarrow(s)|$ and $|I^\uparrow(t)|$ are polynomial in the size of M , we can perform this check in time that is polynomial in the size of M and C . Also, we can compute the local information by defining the \oplus operator

$$I^\downarrow(s) \otimes e = \begin{cases} \{e\} & \text{if } L(s) = i \\ I^\downarrow(s) & \text{otherwise} \end{cases}$$

$$I^\uparrow(t) \otimes e = \begin{cases} \{e\} & \text{if } L(s) = j \\ I^\uparrow(t) & \text{otherwise} \end{cases}$$

with $I^\downarrow(\mathbf{T}) = I^\uparrow(\mathbf{1}) = \emptyset$, and the \oplus operator

$$I \oplus I' = I \cup I'$$

The top-down and bottom-up passes clearly require time that is polynomial in the size of M . The corollary then follows from Theorem [11](#).

4 Specialized Propagators

We now present several MDD propagation algorithms that rely on local information obtained as above. The filtering may not be as strong as for a conventional domain store, in the sense that when specialized to an MDD of width one, it may not remove as many values as a conventional filter would. However, a ‘weak’ filtering algorithm can be very effective when applied to the richer information content of an MDD.

If one prefers not to design a filter specifically for MDDs, there is also the option of using a conventional domain filter by adapting it to MDDs. This can be done in a generic fashion that turns out to be yet another application of the above scheme. Section [4.8](#) explains how this is done.

4.1 Equality and Not-Equal Constraints

We first illustrate MDD propagation of the constraints $x_i = x_j$ and $x_i \neq x_j$. Because these are binary constraints, by Corollary [11](#) the scheme presented in

Section 3.2 achieves MDD consistency in polytime. If we compute I^\downarrow as described in that section, we can achieve MDD consistency for $x_i = x_j$ by deleting an edge $e \in E(s, t)$ whenever (a) $L(s) = j$ and $e \notin I^\downarrow(s)$ or (b) $L(s) = i$ and $e \notin I^\uparrow(t)$. We can achieve MDD consistency for $x_i \neq x_j$ by deleting e whenever (a) $L(s) = j$ and $I^\downarrow(s) = \{e\}$ or (b) $L(s) = i$ and $I^\uparrow(t) = \{e\}$.

We note that this scheme generalizes directly to propagating $f_i(x_i) = f_j(x_j)$ and $f_i(x_i) \neq f_j(x_j)$ for functions f_i and f_j . The scheme can also be applied to constraints $x_i < x_j$. However, in this case we only need to maintain bound information instead of sets of domain values, which leads to an even more efficient implementation.

4.2 Propagating Linear Inequalities

We next focus on the filtering algorithm for general inequalities, as proposed in [1]. That is, we want to propagate an inequality over a separable function of the form:

$$\sum_{j \in J} f_j(x_j) \leq b \tag{1}$$

We can propagate such constraint on an MDD by performing shortest-path computations.

Recall that each edge $e \in E(s, t)$ is identified with a value assigned to $\text{var}(s)$. Supposing that $\text{var}(s) = x_j$, we let the length of edge e be $f_j(e)$ when $j \in J$ and zero otherwise. Thus the length of a **T**-to-**1** path is the left-hand side of (1).

We let $I^\downarrow(s)$ be the length of a shortest path from **T** to s , and $I^\uparrow(s)$ the length of a shortest path from s to **1**. Then we delete an edge $e \in E(s, t)$ when $L(s) \in J$ and every path through e is longer than b ; that is,

$$I^\downarrow(s) + f_{L(s)}(e) + I^\uparrow(t) > b$$

It is easy to compute local information in the form of shortest path lengths, because we can define for $e \in E(s, t)$

$$I^\downarrow(s) \otimes e = \begin{cases} I^\downarrow(s) + f_{L(s)}(e) & \text{if } L(s) \in J \\ I^\downarrow(s) & \text{otherwise} \end{cases}$$

$$I^\uparrow(t) \otimes e = \begin{cases} I^\uparrow(t) + f_{L(s)}(e) & \text{if } L(s) \in J \\ I^\uparrow(t) & \text{otherwise} \end{cases}$$

with $I^\downarrow(\mathbf{T}) = I^\uparrow(\mathbf{1}) = 0$. We also define

$$I \oplus I' = \min\{I, I'\}$$

This inequality propagator achieves MDD consistency as an edge e is always removed unless there exists a feasible solution to the inequality that supports it [1].

4.3 Propagating Two-Sided Inequality Constraints

In this section, we present a generalization of the equality propagator described by Hadzic et al. [3]. It extends the inequality propagator of Section 4.2, but now we store all path lengths instead of only the shortest and longest paths.

Suppose we are given an inequality constraint $l \leq \sum_{j \in J} f_j(x_j) \leq u$, where l and u are numbers such that $l \leq u$. Let $I^\downarrow(s)$ be the set of all path lengths from \mathbf{T} to s , and $I^\uparrow(s)$ the set of all path lengths from s to $\mathbf{1}$. We delete an edge $e \in E(s, t)$ when

$$v + e + v' \notin [l, u], \text{ for all } v \in I^\downarrow(s), v' \in I^\uparrow(t)$$

The local information is computed by defining for $e \in E(s, t)$

$$I^\downarrow(s) \otimes e = \begin{cases} \{v + e \mid v \in I^\downarrow(s)\} & \text{if } L(s) \in J \\ I^\downarrow(s) & \text{otherwise} \end{cases}$$

$$I^\uparrow(t) \otimes e = \begin{cases} \{v + e \mid v \in I^\uparrow(t)\} & \text{if } L(s) \in J \\ I^\uparrow(t) & \text{otherwise} \end{cases}$$

with $I^\downarrow(\mathbf{T}) = I^\uparrow(\mathbf{1}) = \emptyset$. Also $I \oplus I' = I \cup I'$.

When we delete an edge, the information stored at all predecessors and successors becomes ‘stale’, and the information for these nodes must be recomputed to guarantee MDD consistency. However, we will achieve MDD consistency if, every time we delete an edge, we update the node information for all predecessors and successors and repeat this filtering and updating until we reach a fixed point. This follows because the filtering condition above is both necessary and sufficient for an edge to be supported by a feasible solution. Observing that the information can be computed in pseudo-polynomial time, by Theorem 1 this algorithm runs in pseudo-polynomial time (see also [3]).

4.4 Propagating the AllDifferent Constraint

The constraint `AllDifferent`($x_i, i \in J$) requires that the variables x_i for $i \in J$ take pairwise distinct values. We can frame the `AllDifferent` propagator presented in Andersen et al. [1] in terms of our scheme. Let $I^\downarrow(s) = (A^\downarrow(s), S^\downarrow(s))$ and $I^\uparrow(s) = (A^\uparrow(s), S^\uparrow(s))$. Here $A^\downarrow(s)$ is the set of values that appear on all paths from \mathbf{T} to s —that is, the set of values v such that on all \mathbf{T} - s paths, $x_j = v$ for some $j \in J$. $S^\downarrow(s)$ is the set of values v that appear on some \mathbf{T} - s path. $A^\uparrow(s)$ and $S^\uparrow(s)$ are defined similarly.

We can delete edge $e \in E(s, t)$ for $L(s) \in J$ when $e \in A^\downarrow(s) \cup A^\uparrow(t)$. We can also delete e when the variables above s , or the variables below s , form a Hall set. To make this precise, let $X_s^\downarrow = \{x_j \mid j \in J, j < L(s)\}$ be the set of variables in the `AllDifferent` constraint above s , and $X_t^\uparrow = \{x_j \mid j \in J, j > L(s)\}$ the set of variables below s . Then if $|X^\downarrow(s)| = |S^\downarrow(s)|$ (that is, X_s^\downarrow is a Hall set), the values in $S^\downarrow(s)$ cannot be assigned to any variable not in $X^\downarrow(s)$. So we delete e if $e \in S^\downarrow(s)$. Similarly, if $X^\uparrow(t)$ is a Hall set, we delete e if $e \in S^\uparrow(t)$.

Finally, we compute the local information by defining for $e \in E(s, t)$

$$I^\downarrow(s) \otimes e = \begin{cases} I^\downarrow(s) \cup (\{e\}, \{e\}), & \text{if } L(s) \in J \\ I^\downarrow(s), & \text{otherwise} \end{cases}$$

$$I^\uparrow(t) \otimes e = \begin{cases} I^\uparrow(t) \cup (\{e\}, \{e\}), & \text{if } L(s) \in J \\ I^\uparrow(t), & \text{otherwise} \end{cases}$$

where the unions are taken componentwise and $I^\downarrow(\mathbf{T}) = I^\uparrow(\mathbf{1}) = (\emptyset, \emptyset)$. Also we define

$$I \oplus I' = (A \cap A', S \cup S')$$

4.5 Propagating the among Constraint

The **Among** constraint restricts the number of variables that can be assigned a value from a specific subset of domain values. Formally, if $X = (x_1, \dots, x_q)$ is a sequence of variables, S a set of domain values, and ℓ, u, q are constants with $0 \leq \ell \leq u \leq q$, then **Among**(X, S, ℓ, u) requires that

$$\ell \leq |\{i \in \{1, \dots, q\} \mid v_i \in S\}| \leq u$$

We can reduce propagating **Among**(X, S, ℓ, u) to propagating a two-sided separable inequality constraint,

$$\ell \leq \sum_{x_i \in X} f_i(x_i) \leq u,$$

where

$$f_i(v) = \begin{cases} 1, & \text{if } v \in S \\ 0, & \text{otherwise.} \end{cases}$$

Because each $f_i(\cdot) \in \{0, 1\}$, we can compute the information in polynomial time, and by Theorem [1](#) MDD consistency can be achieved in polynomial time for **Among** constraints.

However, this filtering is too slow in practice. Instead, we propose to propagate bounds information instead. That is, we can use the inequality propagator for the pair of inequalities separately, and reason on the shortest and longest path lengths, as in Section [4.2](#)

4.6 Propagating the Element Constraint

We next consider constraints of the form **Element**($x_i, (a_1, \dots, a_m), x_j$), where the a_k are constants. This means that the variable x_j must take the x_i^{th} value in the list (a_1, \dots, a_m) ; that is, $x_j = a_{x_i}$.

Because this is a binary constraint, we can achieve MDD consistency in poly-time by defining $I^\downarrow(s), I^\uparrow(t)$ as in the proof of Corollary [1](#). Supposing that $i < j$, we delete an edge $e \in E(s, t)$ when (a) $L(s) = j$ and $e = a_k$ for no $k \in I^\downarrow(s)$, or (b) $L(s) = i$ and $a_e \notin I^\uparrow(t)$.

4.7 Propagating the Unary Resource Constraint

We consider unary resource constraints of the following form. We wish to schedule a set A of activities on a single resource (non-preemptively). Each activity $a \in A$ has a given release time r_a , deadline d_a , and processing time p_a . We model the problem using variables $X = (x_1, \dots, x_{|A|})$, where $x_i = a$ implies that activity a is the i^{th} activity to consume the resource. That is, we to find the order in which to process the activities, from which the start times can be immediately derived.

First, observe that in our representation the variables encode a permutation of A , which means that we can immediately apply the `AllDifferent` propagator from Section 4.4.

To enforce the time windows, let $I^\downarrow(s)$ be the earliest start time of the activity in position $L(s)$ of the sequence, given the previous activities in the sequence. Let $I^\uparrow(t)$ be latest completion time of the activity in position $L(t) - 1$, given the subsequent activities. Then we can delete edge $e \in E(s, t)$ if the time window is too small to complete activity e ; that is, if

$$\max\{I^\downarrow(s), r_e\} + p_e > \min\{I^\uparrow(t), d_e\}$$

The local information is computed

$$\begin{aligned} I^\downarrow(s) \otimes e &= \max\{I^\downarrow(s) + p_e, r_e + p_e\} \\ I^\uparrow(t) \otimes e &= \min\{I^\uparrow(t) - p_e, d_e - p_e\} \end{aligned}$$

with $I^\downarrow(\mathbf{1}) = -\infty$ and $I^\uparrow(\mathbf{1}) = \infty$. Also $I \oplus I' = (\min\{I^\downarrow, I'^\downarrow\}, \max\{I^\uparrow, I'^\uparrow\})$.

4.8 Using Conventional Domain Filters

An existing domain filter can be adapted to MDD propagation on the basis of local information. However, the resulting propagator may not achieve MDD consistency even when it achieves domain consistency.

The adaptation goes as follows. Following Andersen et al. [11], we define the *induced domain relaxation* $D^\times(M)$ of an MDD M to be a tuple of domains $(D_1^\times(M), \dots, D_n^\times(M))$ where each $D_i^\times(M)$ contains the values that appear on level i of M . That is,

$$D_i^\times(M) = \bigcup_{\substack{s, t \\ L(s) = i = L(t) - 1}} E(s, t)$$

We can perhaps delete edges from a given level i of M by selecting a node s on level i and applying the conventional filter to the domains

$$D_1^\times(M_s), \dots, D_{i-1}^\times(M_s), E^{out}(s), D_{i+1}^\times(M_s), \dots, D_n^\times(M_s) \tag{2}$$

where M_s is the portion of M consisting of all paths through node s . We remove values only from the domain of x_i , that is from $E^{out}(s)$, and delete the corresponding edges from M . This can be done for each node on level i and for each level in turn.

To compute $D^\times(M_s)$, we can regard it as local information $I^\downarrow(s)$ (bottom-up information $I^\uparrow(s)$ is not needed). Then for $e \in E(s, t)$

$$I^\downarrow(s) \otimes e = I^\downarrow(s) \cup (\emptyset, \dots, \emptyset, \{e\}, \emptyset, \dots, \emptyset)$$

where $\{e\}$ is component $L(s)$ of the vector, and the union is taken component-wise. Also $I \oplus I' = I \cup I'$.

This leads to the following lemma.

Lemma 1. *The induced domain relaxation $D^\times(M_s)$ can be computed for all nodes s of a given MDD M in polynomial time (in the size of the MDD).*

Again following Andersen et al. [1], we can strengthen the filtering by noting which values can be deleted from the domains $D_j^\times(M_s)$ for $j \neq i$ when (2) is filtered. If v can be deleted from $D_j^\times(M_s)$, we place the nogood $x_j \neq v$ on each edge in $E^{out}(s)$. Then we move the nogoods on level i toward level j . If $j > i$, for example, we filter (2) for each node on level i and then note which nodes on level $i + 1$ have the property that all incoming edges have the nogood $x_j \neq v$. These nodes propagate the nogood to all their outgoing edges, and so forth until level j is reached, where all edges with nogood $x_j \neq v$ and label v are deleted.

5 Implementation Issues

We have implemented a C++ system for MDD-Based Constraint Programming. For a detailed description of the system, we refer to [7]. We next highlight some of the most important design decisions we have made.

Even though MDDs can grow exponentially large to represent a given constraint perfectly, the basis of MDD-based constraint programming is to control the size of the MDD by specifying a maximum width k . A MDD of width one is equivalent to the conventional domain store, while increasing values of k allow the MDD to converge to a perfect representation of the solution space. An MDD on n variables therefore contains $O(nk)$ nodes. Furthermore, by aggregating the domain values corresponding to multiple (parallel) edges between two nodes, the MDD contains $O(nk^2)$ edges. Therefore, for fixed k , all bottom-up and top-down passes take linear time.

In our system, we do not propagate the constraints until a fixed point. Instead, by default we allocate one bottom-up and top-down pass to each constraint. The bottom-up pass is used to compute the information I^\uparrow . The top-down pass processes the MDD a layer at a time, in which we first compute I^\downarrow , then refine the nodes in the layer, and finally apply the filtering conditions based on I^\uparrow and I^\downarrow .

Our outer search procedure is currently implemented using a priority queue, in which the search nodes are inserted with a specific weight. This allows to easily encode depth-first search or best-first search procedures. Each search tree node contains a copy of the MDD of its parent, together with the associated branching decision. When applying a depth-first search strategy, the total amount of

space required to store all MDDs is polynomial, namely $O(n^2k^2)$. We note that ‘recomputation’ strategies that are common in conventional CP systems, cannot be easily applied in this context, because the MDD may change from parent to child node, due to the node refinement procedure.

6 Experimental Results

In this section, we provide detailed experimental evidence to support the claim that MDD-based constraint programming can be a viable alternative to constraint programming based on the domain store. All the experiments are performed using a 2.33GHz Intel Xeon machine with 8GB memory, using our MDD-based CP solver. For comparison reasons, our solver applies a depth-first search, using a static lexicographic-first variable selection heuristic, and a minimum-value-first value selection heuristic. We vary the maximum width of the MDD, while keeping all other settings the same.

Multiple Among Constraints. We first present experiments on problems consisting of multiple **Among** constraints. Each instance contains 50 (binary) variables, and each **Among** constraint consists of 5 variables chosen at random, from a normal distribution with a uniform-random mean (from [1..50]) and standard deviation $\sigma = 2.5$, modulo 50. As a result, for these **Among** constraints the variable indices are near-consecutive, a pattern encountered in many practical situations. Each **Among** has a fixed lower bound of 2 and upper bound of 3, specifying the number of variables that can take value 1. In our experiments we vary the number of **Among** constraints (from 5 to 200, by steps of 5) in each instance, and we generate 100 instances for each number. We note that these instances exhibit a sharp feasibility phase transition, with a corresponding hardness peak, as the number of constraints increases. We have experimented with several other parameter settings, and we note that the reported results are representative for the other parameter settings; see Hoda [7] for more details.

In Figure 2, we provide a scatter plot of the running times for width 1 versus width 4, 8, and 16, for all instances. Note that this is a log-log plot. Points on the diagonal represent instances for which the running times, respectively number of backtracks, are equal. For points below the diagonal, width 4, 8, or 16 has a smaller search tree, respectively is faster, than width 1, and the opposite holds for points above the diagonal.

We can observe that width 4 already consistently outperforms the domain store, in some cases up to six orders of magnitude in terms of search tree size (backtracks), and up to four orders of magnitude in terms of computation time. For width 8, this behavior is even more consistent, and for width 16, all instances can be solved in under 10 seconds, while the domain store needs hundreds or thousands of seconds for several of these instances.

Nurse Rostering Instances. We next conduct experiments on a set of instances inspired by nurse rostering problems, taken from [8]. The instances are of three different classes, and combine constraints on the minimum and maximum number of working days for sequences of consecutive days of given lengths.

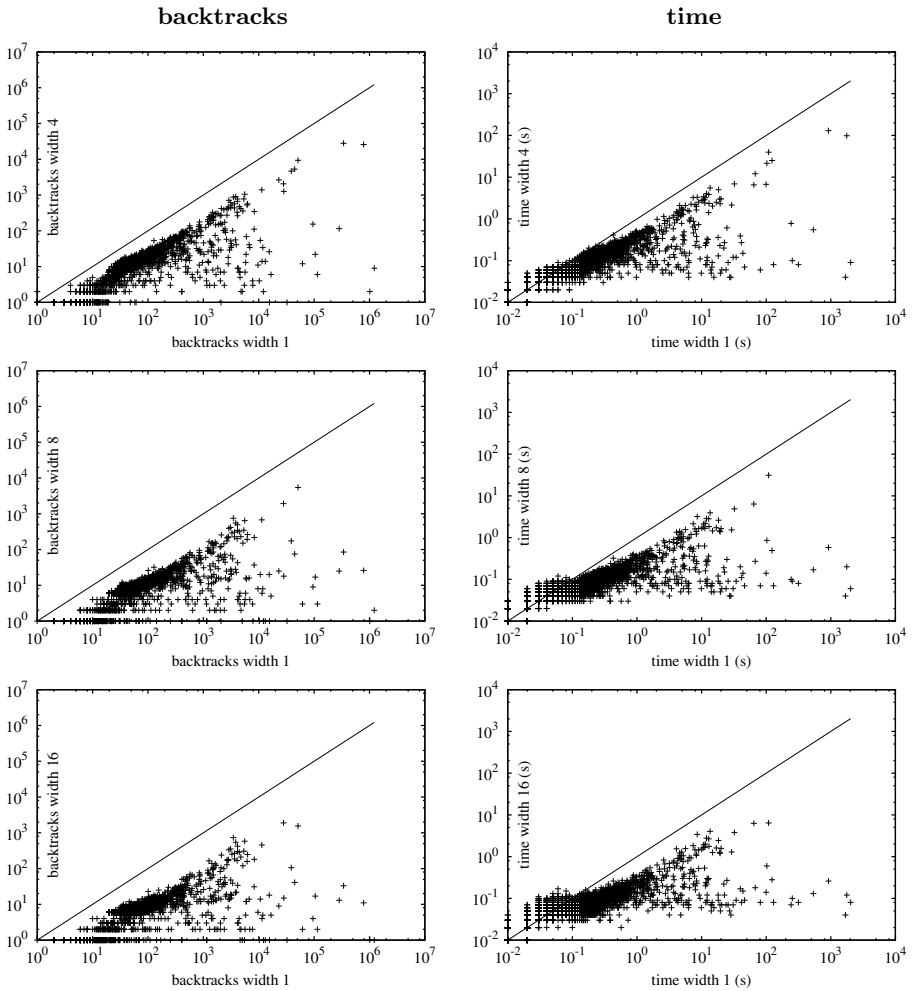


Fig. 2. Scatter plots comparing width 1 versus width 4, 8, and 16 (from top to bottom) in terms of backtracks (left) and computation time in seconds (right) on multiple *Among* problems

That is, class *C-I* demands to work at most 6 out of each 8 consecutive days (max6/8) and at least 22 out of every 30 consecutive days (min22/30). For class *C-II* these numbers are max6/9 and min20/30, and for class *C-III* these numbers are max7/9 and min22/30. In addition, all classes require to work between 4 and 5 days per calendar week. The planning horizon ranges from 40 to 80 days.

The results are presented in Table [11](#). We report the total number of backtracks upon failure (BT) and computation time in seconds (CPU) needed by our MDD solver for finding a first feasible solution, using widths 1, 2, 4, 8, 16, 32, and 64. Again, the MDD of width 1 corresponds to a domain store. For all problem classes we observe a nearly monotonically decreasing sequence of backtracks

Table 1. The effect of the MDD width on time in seconds (CPU) and backtracks (BT) when finding one feasible solution on nurse rostering instances

instance	size	width 1		width 2		width 4		width 8		width 16		width 32		width 64	
		BT	CPU	BT	CPU	BT	CPU	BT	CPU	BT	CPU	BT	CPU	BT	CPU
<i>C-I</i>	40	61,225	55.63	22,443	28.67	8,138	12.64	1,596	3.84	6	0.07	3	0.09	2	0.10
	50	62,700	88.42	20,992	48.82	3,271	12.04	345	2.76	4	0.08	3	0.13	3	0.16
	60	111,024	196.94	38,512	117.66	3,621	19.92	610	6.89	12	0.24	8	0.29	5	0.34
	80	174,417	375.70	64,410	243.75	5,182	37.05	889	12.44	43	0.80	13	0.59	14	0.90
		175,175	442.29	64,969	298.74	5,025	44.63	893	15.70	46	1.17	11	0.72	12	1.01
<i>C-II</i>	40	179,743	173.45	60,121	79.44	17,923	32.59	3,287	7.27	4	0.07	4	0.07	5	0.11
	50	179,743	253.55	73,942	166.99	9,663	38.25	2,556	18.72	4	0.09	3	0.12	3	0.18
	60	179,743	329.72	74,332	223.13	8,761	49.66	1,572	16.82	3	0.13	3	0.18	2	0.24
	70	179,743	391.29	74,332	279.63	8,746	64.80	1,569	22.35	4	0.18	2	0.24	2	0.34
	80	179,743	459.01	74,331	339.57	8,747	80.62	1,577	28.13	3	0.24	2	0.32	2	0.45
<i>C-III</i>	40	91,141	84.43	29,781	38.41	5,148	9.11	4,491	9.26	680	1.23	7	0.18	6	0.13
	50	95,484	136.36	32,471	75.59	2,260	9.51	452	3.86	19	0.43	7	0.24	3	0.20
	60	95,509	173.08	32,963	102.30	2,226	13.32	467	5.47	16	0.50	6	0.28	3	0.24
	70	856,470	1,986.15	420,296	1,382.86	37,564	186.94	5,978	58.12	1,826	20.00	87	3.12	38	2.29
	80	882,640	2,391.01	423,053	1,752.07	33,379	235.17	4,236	65.05	680	14.97	55	3.27	32	2.77

and solution time as we increase the width up to 64. Furthermore, the rate of decrease appears to be exponential in many cases, and again higher widths can yield savings of several orders of magnitude. A typical result (the instance *C-III* on 60 days) shows that where an MDD of width 1 requires 95,509 backtracks and 173.08 seconds of computation time, an MDD of width 32 only requires 6 backtracks and 0.28 seconds of computation time to find a first feasible solution.

7 Conclusion

We have introduced a generic scheme for propagating constraints in MDDs, and showed that all existing MDD-based constraint propagators are instantiations of this scheme. Furthermore, our scheme can be applied to systematically design propagators for other constraints, and we have illustrated this explicitly for the **Among**, **Element**, and unary resource constraints. We further provide experimental results for the first pure MDD-based constraint programming solver, showing that MDD-based constraint programming can yield savings of several orders of magnitude in time and search space as compared to the conventional domain store.

References

- [1] Andersen, H.R., Hadzic, T., Hooker, J.N., Tiedemann, P.: A Constraint Store Based on Multivalued Decision Diagrams. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 118–132. Springer, Heidelberg (2007)
- [2] Cheng, K., Yap, R.: Maintaining Generalized Arc Consistency on Ad Hoc r-Ary Constraints. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 509–523. Springer, Heidelberg (2008)
- [3] Hadzic, T., Hooker, J.N., O’Sullivan, B., Tiedemann, P.: Approximate Compilation of Constraints into Multivalued Decision Diagrams. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 448–462. Springer, Heidelberg (2008)

- [4] Hadzic, T., Hooker, J.N., Tiedemann, P.: Propagating Separable Equalities in an MDD Store. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 318–322. Springer, Heidelberg (2008)
- [5] Hadzic, T., O’Mahony, E., O’Sullivan, B., Sellmann, M.: Enhanced Inference for the Market Split Problem. In: Proceedings of ICTAI, pp. 716–723. IEEE, Los Alamitos (2009)
- [6] Hawkins, P., Lagoon, V., Stuckey, P.J.: Solving Set Constraint Satisfaction Problems Using ROBDDs. *JAIR* 24(1), 109–156 (2005)
- [7] Hoda, S.: Essays on Equilibrium Computation, MDD-based Constraint Programming and Scheduling. PhD thesis, Carnegie Mellon University (2010)
- [8] van Hoeve, W.-J., Pesant, G., Rousseau, L.-M., Sabharwal, A.: New Filtering Algorithms for Combinations of Among Constraints. *Constraints* 14, 273–292 (2009)

A Complete Multi-valued SAT Solver

Siddhartha Jain¹, Eoin O'Mahony², and Meinolf Sellmann^{1,*}

¹ Brown University, Department of Computer Science
P.O. Box 1910, Providence, RI 02912, U.S.A.

{sj10, sello}@cs.brown.edu

² Cork Constraint Computation Centre
University College Cork, Cork, Ireland
e.omahony@4c.ucc.ie

Abstract. We present a new complete multi-valued SAT solver, based on current state-of-the-art SAT technology. It features watched literal propagation and conflict driven clause learning. We combine this technology with state-of-the-art CP methods for branching and introduce quantitative supports which augment the watched literal scheme with a watched domain size scheme. Most importantly, we adapt SAT nogood learning for the multi-valued case and demonstrate that exploiting the knowledge that each variable must take exactly one out of many values can lead to much stronger nogoods. Experimental results assess the benefits of these contributions and show that solving multi-valued SAT directly often works better than reducing multi-valued constraint problems to SAT.

1 Multi-valued SAT

One of the very successful solvers for constraint satisfaction problems (CSPs) is Sugar [23]. It is based on the reduction of CSP to the satisfiability problem (SAT). Sugar first encodes the given problem as a SAT formula and then employs MiniSAT [7] to solve the instance. Somewhat surprisingly, Sugar won the ACP global constraint competition in the past two years [21]. Our work is highly motivated by the success of this solver. Our objective is to provide a solver which could replace MiniSAT for reduction-based CSP solvers and work with even better efficiency by taking into account that CSP variables usually have non-boolean domains. To this end, let us begin by formally defining the multi-valued SAT problem.

Definition 1 (Multi-Valued Variables). A multi-valued variable X_i is a variable that takes values in a finite set D_i called the domain of X_i .

Definition 2 (Multi-Valued Clauses)

- Given a multi-valued variable X_i and a value v , we call the constraint $X_i = v$ a variable equation on X_i . A variable equation $X_i = v$ is called satisfiable iff $v \in D_i$.
- Given a set of multi-valued variables $X = \{X_1, \dots, X_n\}$ and a set $T \subseteq X$, a clause over T is a disjunction of variable equations on variables in X .

* This work was supported by the National Science Foundation through the Career: Cornflower Project (award number 0644113).

Example 1. Given variables X_1, X_2, X_3 with domains $D_1 = \{1, \dots, 5\}$, $D_2 = \{\text{true}, \text{false}\}$, and $D_3 = \{\text{red}, \text{green}, \text{blue}\}$,

$$(X_1 = 1 \vee X_1 = 3 \vee X_1 = 4 \vee X_2 = \text{false} \vee X_3 = \text{red}) \quad (1)$$

is a multi-valued clause over X_1, X_2, X_3 .

Note that multi-valued clauses have no negation. A classic SAT clause would be written as $(X_1 = \text{false} \vee X_2 = \text{true} \vee X_3 = \text{false})$ in this notation. To save memory, it can be very helpful to encode large sets of allowed values by specifying the disallowed values only. E.g., in the example above we may prefer to write

$$((X_1 \neq 2 \wedge X_1 \neq 5) \vee X_2 = \text{false} \vee X_3 = \text{red}). \quad (2)$$

While the solver that we developed for this paper allows these inputs, they are not part of the problem definition itself to keep the theory free from confusing implementation details.

Definition 3 ((Partial) Assignments and Feasibility). *Given a set of multi-valued variables $X = \{X_1, \dots, X_n\}$, denote with D the union of all domains D_1, \dots, D_n . Furthermore, denote with S, T subsets of X .*

- A function $\alpha : S \rightarrow D$ is called an assignment of variables in S . α is called partial iff $|S| < n$, and complete otherwise. If $|S| = 1$, α is called a variable assignment.
- An assignment α is called admissible iff $\alpha(X_i) \in D_i$ for all $X_i \in S$.
- An assignment α of variables in S is called feasible with respect to a clause over T iff $T \setminus S \neq \emptyset$ or if there exists a variable equation $X_i = \alpha(X_i)$ in the clause.
- Given a clause c , a complete, admissible, and feasible assignment is called a solution for c .

Definition 4 (Multi-Valued SAT Problem). *Given a set X of multi-valued variables and a set of clauses over subsets of X , the multi-valued SAT problem (MV-SAT) is to decide whether there exists an assignment α that is a solution for all clauses.*

As MV-SAT allows boolean variables, the problem is at least as hard as SAT and is thus NP-hard. On the other hand, using standard CP encodings (direct encoding, support encoding, or order encoding [24,10,23]), we also know that MV-SAT can be reduced to SAT which makes the problem NP-complete.

Although the problems are equivalent in complexity (as all NP-complete problems are) and also very similar in structure, we consider MV-SAT an interesting generalization of SAT. In [1] it was shown that many problems which are currently being solved as SAT problems do in fact model problems with multi-valued variables. Moreover, many applications, especially in verification, naturally exhibit multi-valued variables. The purpose of this paper is to show that handling multi-valued variables directly rather than encoding them by means of boolean variables can lead to substantial improvements in computational efficiency.

2 Efficient Incremental Clause Filtering

The main inference mechanism in CP and SAT is constraint filtering. That is, for a given constraint we want to identify those variable assignments that cannot be extended to a solution for a given constraint and the current domains. The corresponding domain values are then removed and the process is iterated until no constraint can remove further domain values.

2.1 Unit Propagation in SAT

A classic SAT clause only ever removes a value (true or false) from a variable domain when all other variables in the clause have been assigned a value which does not satisfy the clause. In this case, we speak of a *unit clause* and the process of filtering clauses in this way is called *unit propagation*.

An elegant way of incrementally performing the filtering of SAT clauses was proposed in [18]. Each clause watches only two variable equations. In SAT, these are commonly referred to as *literals*. As long as both watched literals can be satisfied, no filtering can take place. When one watched literal cannot be satisfied anymore, we search for another second literal that can still be satisfied. Only when no second satisfiable literal can be found is the clause unit and we commit the only remaining variable assignment that can still satisfy the clause.

The beauty of this way of performing unit propagation is that it is not necessary to traverse all clauses that involve any variable that has been set to a value. In essence, the two watched literals give us an effective pre-check whether a clause can filter any values. Only when this pre-check fails, i.e. when one of the watched literals is affected, we need to perform any work. Otherwise the cost is not even constant per unaffected constraint, there is in fact no work to do for them at all. This last fact is key for handling problem instances with many constraints and for learning a large number of redundant clauses during search.

Note that we always start the search for a replacement watched literal at the old lost literal and then wrap around if the tail of the clause did not contain a valid support. This prevents us from looking at the same lost supports over and over again which would happen if we always started the search at the first literal in the clause. The method guarantees that each clause literal will be looked at most thrice on any path from the root to any leaf in the search tree. Note further that this scheme is very backtrack friendly as all non-unit constraints do not need to update their watched literals upon backtracking as their current supports are obviously still valid for any ancestor node in the search tree.

2.2 Watched Variable Equations

For multi-valued clauses we can use the very same approach. Note however that multi-valued clauses can trigger filtering earlier. As soon as all variable equations that can still be satisfied regard just one variable, all domain values of this variable that do not satisfy the clause can be removed from its domain. For example, consider the clause from earlier: $(X_1 = 1 \vee X_1 = 3 \vee X_1 = 4 \vee X_2 = \text{false} \vee X_3 = \text{red})$. As soon as the domains of X_2, X_3 do not contain the values false and red anymore, respectively, values 2 and 5 can be removed from D_1 .

To accommodate this fact we only need to ensure is that the two variable equations that are watched cannot regard the same variable. This is automatically guaranteed in SAT when we remove tautologies upfront. Although as we have seen earlier; multi-valued clauses can have a number of variable equations in the same variable.

The two watched variable equations approach therefore watches two variable equations that regard two different variables. To this end, in each clause we group the variable equations according to their respective variables, and we fix an ordering of the variables and their allowed values in the clause. When a watched variable equality is

affected, we search for a replacement starting at the old variable and its old value. If that does not lead to a replacement support, we continue with the next variable, whereby we skip over the variable that is used for the second variable equality.

2.3 Quantitative Supports

In multi-valued SAT we can enhance this scheme by allowing a different kind of support that is based on the size of the current domain of a variable. For a given clause c , let us denote with i_c the number of variable equations that regard X_i in c . Furthermore, let us denote with d_i the size of the initial domain of X_i . The observation is that there must exist at least one satisfiable variable equality for a given variable as long as the size of the current domain is still bigger than the number of values that are disallowed by this clause when all variable equations regarding other variables in this clause cannot be satisfied. Formally: $|D_i| > d_i - i_c \Rightarrow \exists v \in D_i : X_i = v \in c$.

Example 2. Given are two variables X_1, X_2 with initial domains $D_1 = D_2 = \{1, \dots, 6\}$, and a clause $c = (X_1 = 1 \vee X_1 = 2 \vee X_1 = 3 \vee X_1 = 4 \vee X_1 = 5 \vee X_2 = 1)$. The constraint just says that $X_2 = 1$ or $X_1 \neq 6$. Now assume that the current domain of X_1 is $D_1 = \{2, 3, 6\}$. The clause only disallows one value for X_1 , but three values are still allowed: $|D_1| = 3 > 1 = 6 - 5 = d_1 - i_c$. Consequently, there must still exist a variable equality on X_1 in c that can still be satisfied, in our case e.g. $X_1 = 2$.

The point of these quantitative supports is that we do not need to place a bet on any particular value for a given variable. Instead, a clause only needs to be looked at when the domain size of a watched variable falls below or equal the threshold $d_i - i_c$, no matter which particular values are lost until this happens. Especially when few values are disallowed for a variable, watching quantitative supports can save us a lot of work. Note that this type of support is not actually new in constraint programming. Solvers like IBM CP Solver have long associated the filtering of constraints with certain events, such as the event that a variable is bound (i.e. when its corresponding domain has shrunk to size 1). The only difference here is that we allow multi-valued clauses to set their own specific variable domain threshold which triggers when they need to be queued for filtering again.

3 Learning Nogoods for Multi-valued SAT

We have seen in the previous section that multi-valued clauses offer the potential for filtering even before the number of satisfiable variable equations is 1, as long as the ones that remain satisfiable all constrain the same variable. Depending on the SAT encoding, a boolean SAT solver that learns redundant constraints can improve its filtering effectiveness to achieve the same, but the fact that this filtering is guaranteed is already an advantage of modeling a problem as multi-valued SAT over classic SAT.

A second and probably more important advantage of multi-valued SAT is that we can learn better implied constraints by exploiting our knowledge that each variable must take exactly one value. Before we explain how this can be achieved, let us begin by reviewing conflict driven clause learning in SAT.

3.1 Conflict Analysis

Unit propagation is an incomplete inference mechanism in so far as it does not guarantee that all variable domains have size 1 at the end and that it is not guaranteed that a variable domain will be empty even when the given formula has no solution. The only facts that we know for sure are that variables that are unit must be set to the corresponding value in any solution and, consequently, that the formula has no solution if unit propagation finds a variable that can neither be set to true nor to false.

Due to this incompleteness of our filtering method we need to conduct a search for a solution. We assign variables one after the other to values in their domain, whereby after each assignment we perform unit propagation to simplify the formula or to detect that our current partial assignment cannot be extended to a solution. In the latter case, we speak of a *failure* and the variable that can neither be true nor false is called a *conflict variable*.

At this point, we could just undo the last variable assignment and try a different one. We can do much better, though. The power of modern SAT solvers lies in their ability to analyze the cause of the failure and to construct a redundant constraint, a so-called *nogood*, that will help prevent us from making the same error again. Crucial for this conflict analysis is our ability to give the exact reasons why a constraint filters a value. SAT clauses are very well suited for conflict analysis as we can trace exactly which prior domain reductions triggered the filtering.

In particular, modern systematic SAT solvers set up a so-called *implication graph*. It allows us to trace back the reasons why certain domain values have been removed. By collecting all causes for the removal of all domain values of the conflict variable, we can state a new constraint which forbids that a sufficient condition for a failure occurs. The easiest way to explain how the implication graph is set up and used is by means of an example which we will use throughout this section.

Example 3. Consider a constraint satisfaction problem (CSP) with five variables and five constraints. The variables have domains $D_1 = D_2 = D_4 = \{1, 2, 3\}$, $D_3 = \{1, \dots, 6\}$, $D_5 = \{1, 2\}$. The constraints are $X_1 \neq X_4$, $X_2 \neq X_3$, $X_3 + 9 \geq 5X_4$, $X_3 + 4 \geq 5X_5$, and $X_4 \neq X_5 + 1$.

Say we model this problem by introducing boolean variables x_{ij} which are true iff $X_i = j$. To ensure that each CSP variable takes exactly one value we introduce clauses $(\bigvee_j x_{ij} = \text{true})$ for all i , and $(x_{ij} = \text{false} \vee x_{ik} = \text{false})$ for all i and $j < k$. In particular, we have a clause

$$(x_{31} = \text{false} \vee x_{36} = \text{false}). \quad (3)$$

We need to add multiple clauses for each CSP constraint. Among others, for constraints $X_1 \neq X_4$, $X_2 \neq X_3$, and $X_4 \neq X_5 + 1$ we add

$$(x_{11} = \text{false} \vee x_{41} = \text{false}) \wedge (x_{22} = \text{false} \vee x_{32} = \text{false}) \wedge (x_{42} = \text{false} \vee x_{51} = \text{false}). \quad (4)$$

Constraints $X_3 + 9 \geq 5X_4$ and $X_3 + 4 \geq 5X_5$ are enforced by the clauses

$$(x_{41} = \text{true} \vee x_{42} = \text{true} \vee x_{36} = \text{true}) \wedge (x_{51} = \text{true} \vee x_{36} = \text{true}). \quad (5)$$

The resulting SAT formula has no unit clauses, and so we begin our search. Assume that we first commit $x_{11} \leftarrow \text{true}$. To record this setting, we add a node $(x_{11} \neq \text{false})$

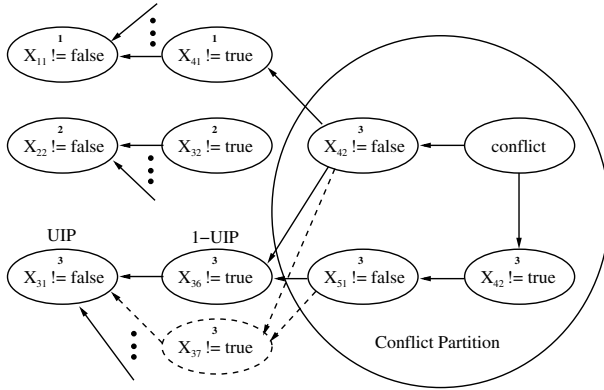


Fig. 1. SAT implication graph for Example 3. The solid nodes and arcs depict the relevant part of the implication graph when $D_3 = \{1, \dots, 6\}$. The dashed nodes and arcs are part of the implication graph when $D_3 = \{1, \dots, 7\}$.

to our implication graph¹ and we note that this node is added at search depth level 1 (compare with Figure 1 and ignore the dashed nodes and arcs). Among the clauses that we consider for this example, only the first clause in (4) becomes unit, and we infer $(x_{41} \neq \text{true})$. We add also this node to the graph, mark it with depth level 1, and draw an arrow from the second to the first node to record that the first variable inequality implies the second.

After unit propagation is complete, we may next set $x_{22} \leftarrow \text{true}$. Again, we add a node $(x_{22} \neq \text{false})$ to our graph and note that it was added on depth level 2. By (4) we infer $(x_{32} \neq \text{true})$, add this node with depth mark 2, and add an arc to $(x_{22} \neq \text{false})$.

Again after all unit propagation is complete, let us assume that we now commit $x_{31} \leftarrow \text{true}$. Again, we add $(x_{31} \neq \text{false})$, and by (3) we infer $(x_{36} \neq \text{true})$ (both at depth level 3, just like all nodes that follow). The first clause in (5) is now unit and we add a new node $(x_{42} \neq \text{false})$. We add arcs to node $(x_{41} \neq \text{true})$ and to node $(x_{36} \neq \text{true})$, as both variable inequalities are needed to make this implication. The second clause in (5) is also unit, and we infer $(x_{51} \neq \text{false})$ which is implied by $(x_{36} \neq \text{true})$. The last clause in (4) is now unit and we add node $(x_{42} \neq \text{true})$. Together with the earlier implied $(x_{42} \neq \text{false})$ we have now reached a conflict. In Figure 1 we show the implication graph at this point (please ignore the dashed elements).

This graph is used to compute nogoods: Any set of nodes that represents a cut between the conflict node on one side and all branch nodes (i.e. nodes without outgoing arcs) on the other defines a valid constraint. For example, all paths from the conflict node to any branch node must visit either $(x_{42} \neq \text{false})$ or $(x_{36} \neq \text{true})$. Consequently, it is sound to enforce that not both variable inequalities hold at the same time or, equivalently, that $(x_{42} = \text{false} \vee x_{36} = \text{true})$. Conveniently, this constraint is again a clause! In terms of our CSP variables means that $X_4 = 2$ implies $X_3 = 6$.

¹ Note that our notation is not standard in SAT. We use it here because it will naturally generalize to multi-valued SAT later.

Another cut set is the set of all reachable branch nodes, in our case $(x_{11} \neq \text{false})$ and $(x_{31} \neq \text{false})$. This cut results in the clause $(x_{11} = \text{false})$ and $(x_{31} = \text{false})$, or $X_1 = 1$ implies $X_3 \neq 1$. This latter fact is interesting as it obviously means that after our first search decision we could already have inferred $x_{31} \neq \text{true}$. Another way to put this is to say that the second branching decision was irrelevant for the conflict encountered, a fact that we can exploit to undo multiple search decisions, a process which is commonly referred to as *back-jumping* or *non-chronological backtracking*.

In order to achieve immediate additional propagation by the newly learned clause after back-jumping, we need to make sure that the clause contains only *one* node from the last depth level. Then and only then the newly learned clause will be unit and thus trigger more filtering after back-jumping. To find nodes that can be extended to a full cut using only nodes from lower depth levels, we consider only paths between the conflict node and the last branching decision. In Figure 1 this happens to be the subgraph induced by the nodes marked with depth level 3 (in general it would be a subset of the nodes on the lowest level). In this subgraph, we search for cut points, i.e., nodes that all paths from conflict to branch node must visit. In SAT these are commonly referred to as *unit implication points (UIP)*.

Note that these UIPs can be computed in time linear in the number of edges of the subgraph. In our case, there are two UIPs, $(x_{36} \neq \text{true})$ and the branch node itself, $(x_{31} \neq \text{false})$. In [25] it was established that it is beneficial to consider the UIP that is closest to the conflict. This is called the *first UIP (1-UIP)*. Now, we group all nodes between the 1-UIP and the conflict together (marked by the circle in Figure 1). The cut set is then the set of all nodes that have a direct parent in this set. In our case, these are $(x_{36} \neq \text{true})$ and $(x_{41} \neq \text{true})$, which gives the nogood $(x_{36} = \text{true} \vee x_{41} = \text{true})$; or, in terms of the CSP variables, $X_4 \neq 1$ implies $X_3 = 6$.

3.2 Unit Implication Variables

As the example shows, conflict analysis can result in powerful inference. We will now show how we can learn even stronger nogoods in multi-valued SAT.

Consider again Example 3 but this time let us assume that $D_3 = \{1, \dots, 7\}$. In our SAT model, the clauses enforcing $X_3 + 9 \geq 5X_4$ and $X_3 + 4 \geq 5X_5$ change to $(x_{41} = \text{true} \vee x_{42} = \text{true} \vee x_{36} = \text{true} \vee x_{37} = \text{true})$ and $(x_{51} = \text{true} \vee x_{36} = \text{true} \vee x_{37} = \text{true})$.

If we branch as before, the implication graph at depth level 3 includes the dashed arcs and nodes in Figure 2. We observe that it has only one UIP now, and that is the branch node itself. Consequently, the nogood learned is much weaker, we only

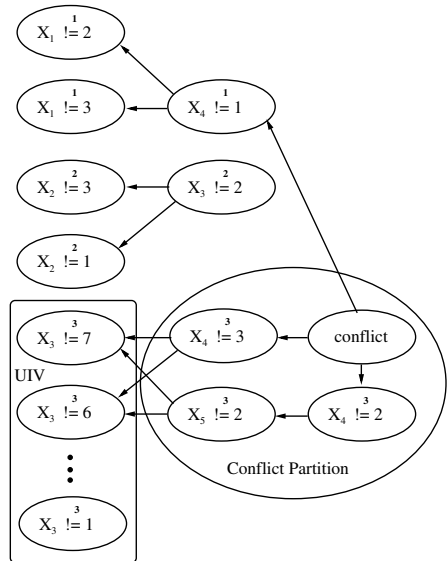


Fig. 2. MV-SAT Implication Graph

infer $(x_{31} = \text{false} \vee x_{41} = \text{true})$; or, in terms of the CSP variables, that $X_4 \neq 1$ implies $X_3 \neq 1$.

In Figure 2 we show the implication graph for the corresponding multi-valued SAT model. Note that this graph no longer contains a single node that corresponds to the branching assignment. This is rather given as a number of variable inequalities. Observe further that in this graph there does not exist a UIP at all anymore. However, recall from Section 2 that a multi-valued clause can cause filtering even when multiple variable equations are still satisfiable, as long as all of them regard the same variable. For us, this means that we no longer need to find a cut point, but potentially an entire set of nodes:

Definition 5 (Unit Implication Variable). *Given a multi-valued SAT problem and an implication graph G , assume that, if all nodes associated with variable X are removed from G , there exists no path from the conflict node to any branch node on the lowest branch level. Then, we call X a unit implication variable (UIV).*

In our example, X_3 is a UIV. Based on its associated cut set, we can again compute a conflict partition and set the nogood as the negation of the conjunction of all variable inequalities which have a direct parent in the conflict partition. In our example, we thus find the multi-valued clause $(X_4 = 1 \vee X_3 = 6 \vee X_3 = 7)$. After backjumping to the highest depth level after the conflict level in our learned clause (in our case level 1), this clause is unit and prunes the domain of X_3 to $D_3 = \{6, 7\}$. That is, equipped with this nogood the solver does not need to branch on $X_3 \leftarrow 2$, $X_3 \leftarrow 3$, $X_3 \leftarrow 4$, and $X_3 \leftarrow 5$ as all of these settings would fail for the very same reason as $X_3 \leftarrow 1$ did.

The challenge here is to find UIVs efficiently. Unfortunately this task no longer consists of the trivial linear computation of a cut point. We propose the following approach. First, we compute the shortest (in the number of nodes) path from any conflict variable node to any branch node. Only variables associated with nodes on this path can be UIVs. We call this set of variables the *candidate set*. Next we update the costs of the edges in the graph such that visiting a node associated with a variable in the candidate set incurs a cost of one, while all other nodes cost nothing. We compute another shortest path based on this cost function. Again, only variables associated with nodes on this path can be UIVs. We can therefore potentially reduce the candidate set. We repeat this process as long as the candidate set keeps shrinking. Finally, we test each remaining candidate by incurring a cost of one for nodes associated with the candidate variable only. If and only if the cost of the shortest path is greater than zero, then this implies that every path from a conflict node to a branch node must pass through a node associated with the candidate variable, which is therefore a UIV. It follows:

Lemma 1. *The set of all unit implication variables can be computed in time $O(mn)$, where m is the number of edges and n is the number of nodes in the subgraph of paths between a conflict and a branch nodes.*

Proof. Apart from the first and one other which establishes that the candidate set does not shrink anymore, each shortest path computation reduces the candidate set by at least one candidate. As there are at most n candidates, we require at most $n + 2$ shortest path computations. The implication graph is a directed acyclic graph, and therefore each shortest path computation takes time $\Theta(m)$. \square

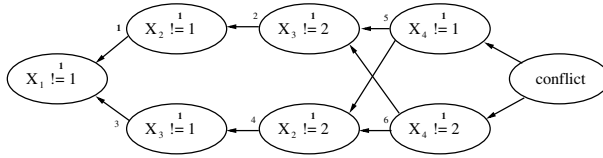


Fig. 3. Implication graph for a problem with four variables with domains $D_1 = D_4 = \{1, 2\}$ and $D_2 = D_3 = \{1, 2, 3\}$

While it is reassuring that UIVs can be computed in polynomial time, a time bound of $O(mn)$ is certainly not very appealing seeing that we need to compute a nogood at every failure. In practice we can of course hope that way fewer than n shortest path computations will be necessary. Fortunately, as our experiments will show, this hope is very well justified.

3.3 Non-dominated UIVs

Recall that in SAT, if there are many UIPs, we choose the one that is closest to the conflict. In multi-valued SAT, we may also have multiple UIVs, whereby the last branching variable is always one of them. The question arises which UIV we should prefer. It is easy to see that there is no longer one unique UIV that dominates every other. For example, see the implication graph in Figure 3

Definition 6 (Non-Dominated UIV). We call a UIV non-dominated if there exists a path from the conflict to a branch node where a node associated with the UIV is the first node on the path that is associated with a UIV.

In our algorithm, we can easily compute a non-dominated UIV by testing the remaining candidates in the order in which they appear on the first shortest path that we computed. Our solver learns the nogood that corresponds to this UIV.

An important aspect of our computing non-dominated UIVs is that they give us a good indication of the strength of the nogoods that we compute. In our experiments, we found that on problems where the vast majority of UIVs coincides with the branching variable, learning sophisticated nogoods is often a waste of time. Consequently, when our solver detects that the number of UIVs that are different from the branching variable drops below 5%, we no longer attempt to find improved, non-dominated UIVs and simply use the branching variable to define the next nogood instead.

3.4 Nogood Management

Among others, our solver offers impacts for selecting the branching variable [17]. Impacts measure the reduction in search space size achieved by propagation after each branching step. The concept has been found very effective for selecting branching variables.

As we learn more and more nogoods through the course of the search, an important task for any conflict driven solver is the management of learned constraints. First, due to limited memory it is simply not feasible to store all learned nogoods until an instance

is solved. Second, for the efficiency of the solver it is essential that we forget redundant constraints which only cause work but rarely filter anything.

We use impacts to determine nogoods that can be deleted without losing much inference power. Whenever a constraint filters some values during the propagation of the effects of a branching decision, we associate the constraint with the *entire* reduction in search space that all constraints achieve together. The rationale behind looking at the entire reduction is that a constraint may not remove many values but very important ones which trigger lots of follow-up propagation.

We keep a running average of these reductions for each learned constraint. When the number of learned clauses reaches a limit (which grows over time), we remove roughly half of the learned clauses by removing all that have an average reduction below the median of all learned clauses. To ensure the completeness of the approach, clauses are protected from removal when they are currently unit.

This scheme works well in principle, but it has one major drawback: constraints which are part of some high impact propagation and which then never become unit again would clog up our system as their average reduction stays high. Therefore, in regular intervals we decrease the expected reduction for each learned constraint by a certain percentage. In our experiments, we decrease the expected reduction by 7% every 100 failures. In this way, constraints that have not been useful in a while get discounted.

4 Related Work

There are many approaches which reduce CSP to SAT and then employ a standard boolean SAT solver. A number of different encodings have been proposed for this purpose [24][10][23]. In the award-winning paper from Ohrimenko et al. [15], CSP propagators themselves were encoded lazily as SAT clauses which gave very good results on scheduling problems. The CSP solver Sugar [23], which won the ACP global constraints competition in the past two years, computes very efficient encodings for various global constraints and then employs MiniSAT [7] to solve the resulting SAT problem. Our work is heavily influenced by these studies and have motivated us to provide a back-end SAT solver that could directly exploit the fact that variables must take exactly one out of many values.

In [11][14], classical CSP nogoods [19] were generalized to accommodate multi-valued variables better. Pioneering work on multi-valued SAT was presented in [23][4]. Here, the then state-of-the-art SAT solvers Chaff [18] and SATZ [12] were augmented with domain-based branching heuristics. Very good speed-ups over the performance over baseline SAT solvers were reported which were solely due to the ability of the multi-valued solver to take domain sizes into account when branching.

An incomplete multi-valued SAT solver was presented in [8]. It is based on an adaptation of the well known local search solver WalkSAT [20]. It was shown that working the knowledge that each variable must take exactly one out of many values into the solver can lead to superior performance on instances from various problem classes with larger variable domain sizes.

The only “pure” complete multi-valued SAT solver we know of was presented in [13]. It was named CAMA and like our own solver it features propagation based on watched

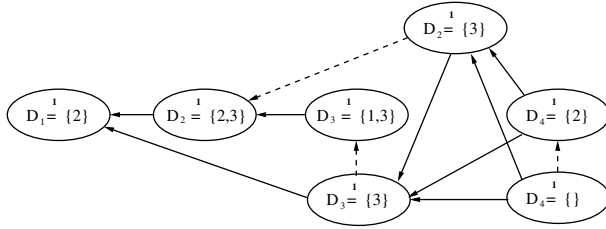


Fig. 4. CAMA implication graph for the same example as in Figure 3

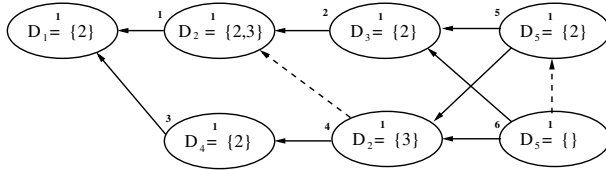


Fig. 5. CAMA implication graph for a problem with five variables with domains $D_1 = D_3 = D_4 = D_5 = \{1, 2\}$ and $D_2 = \{1, 2, 3\}$

literals (albeit without quantitative supports) and a nogood learning method which exploits the knowledge about multi-valued variables. Like us the authors attempt to learn improved nogoods.

In CAMA, a nogood is not constructed through the analysis of an implication graph but through resolution. An implication graph is used only for the computation of a UIV. The implication graph differs considerably from ours, though. As we will see, due to its structure, CAMA is not able to identify non-dominated UIVs, for two reasons:

First, CAMA does not consider pure variable inequalities for learning nogoods, but the entire domain of each variable after a value has been removed. The current domain, however, reflects *all* domain reductions on the variable and not just the ones that are relevant for the filtering that is triggered. Consequently, CAMA needs to trace back the relevant domain reductions, and since it conservatively assumes that the relevant domain reductions happened earlier during propagation, it may miss a unit implication point. In Figure 4 we show a CAMA implication graph for the same example as depicted in Figure 3 where we also mark the order in which the nodes are added to the graph. CAMA computes, in linear time, the cut point in this graph that is closest to the conflict node. As we can see, due to the dashed edges which are needed to denote the implications by earlier domain reductions on the same variable, the only cut point is the branching node itself. CAMA thus finds variable X_1 as UIV which is dominated both by X_2 and X_3 as we can easily see in Figure 3.

The second reason why CAMA cannot identify non-dominated UIVs is that it is simply unable to identify all UIVs by only considering cut points in its implication graph. In Figure 5 we show a different example. Here, even when we ignore the dashed arcs, the only cut point is the branch node, and CAMA chooses X_1 as UIV which is

dominated by X_2 . In summary, CAMA’s nogood learning method runs in linear time, but therefore the nogoods found are in general not as strong as they could be.

5 Numerical Results

We have introduced quantitative supports and non-dominated UIVs for nogood learning in multi-valued SAT. We will now study these contributions and finally compare our solver with standard SAT technology.

5.1 Benchmark Sets and Architecture

For our experiments, we use the following four classes of problems: quasi-group with holes, random binary constraint satisfaction problems, n-queens, and graph coloring.

The quasi-group with holes instances were produced by the generator of Carla Gomes. Instances of different sizes and different percentages of holes were used (40% and 42% which is close and right at the phase transition). Ten instances were generated for each parameter setting of the generator and collected in a set named qwh-[order]-[percent holes].

Random binary constraint satisfaction problems were generated by the generator of Christian Bessiere available at [5]. Instances vary in number of variables, domain size, number of constraints, and constraint tightness. We fix the density of the constraint graph at 0.5 and then derive the value for the critical constraint tightness using the formula given in [16] which is the value for which the BCSP problems are generally hard. We then generate instances with constraint tightness slightly above and below the critical value. Ten instances were generated for each parameter setting of the generator and collected in a set named b-[vars]-[vals]-[tightness].

The n-queens model consists of the standard three types of all different constraints; one enforcing that queens cannot attack each other on the columns, and two enforcing that the queens cannot attack each other on diagonals. In CMV-SAT-1 the all different constraint is decomposed into a clique of not equal constraints. Not equal constraints are transformed into disjunction of variable assignments. The SAT encoding of all different constraints provided by Sugar is described in [22].

The graph coloring instances are part of the DIMACS standard [9]. The problems were changed into decision problems as opposed to optimization problems by setting the desired number of colors to the best known value. We use the subset of 44 instances which could be solved in under one hour of CPU time.

For each instance, we report the average statistics (runtime, nodes, failures, etc.). For all experiments including the ones on different configurations of CMV-SAT-1 we used ten different seeds per instance and ran on Intel Core 2 Quad Q6600 processors with 3GB of RAM.

5.2 Quantitative Supports

In Table 1 we give the average time per choice point when using and when not using quantitative supports. We see clearly that quantitative supports speed up the propagation process considerably and almost independently of the type of problem that is solved.

Table 1. Time [ms] per choice point when using (+Q) and not using (-Q) quantitative supports

QWH	+Q		-Q		BCSP	+Q		-Q		GraphCol	+Q		-Q	
qwh-25-40	3.91	5.44	b-25-20-.43	0.70	0.90	fpsol2.i.1	1.14	1.74						
qwh-25-42	3.04	4.18	b-25-25-.45	0.92	1.29	inithx.i.1	0.87	1.29						
qwh-27-40	3.96	5.90	b-25-30-.47	1.15	1.60	inithx.i.2	0.38	0.62						
qwh-27-42	3.21	5.06	b-25-40-.5	1.45	1.93	le450_15a	0.54	0.67						
qwh-29-40	4.56	6.67	b-30-20-.37	0.75	0.96	le450_15b	0.36	0.49						
qwh-29-42	3.55	4.48	b-30-25-.39	0.94	1.25	le450_25a	0.24	0.40						
qwh-31-40	4.19	5.69	b-30-30-.33	1.05	1.36	le450_25b	0.24	0.38						
qwh-31-42	3.29	5.07	b-30-40-.35	1.66	2.19	le450_5a	0.45	0.72						
qwh-33-40	4.31	6.39	b-35-20-.26	0.82	1.02	le450_5c	1.95	2.70						
qwh-33-42	3.71	5.32	b-35-25-.28	1.17	1.46	miles1500	2.23	3.52						
qwh-35-40	5.33	6.79	b-35-30-.29	1.48	2.00	queen8_8	0.52	0.53						
qwh-35-42	4.21	5.44				queen9_9	1.03	1.03						

Table 2. Comparison between non-dominated (ND-UIV) and quick UIVs (Q-UIV). Time in [s].

QWH	ND-UIV			Q-UIV			GraphCol	ND-UIV			Q-UIV		
	Paths	Time	Fails	Time	Fails			Paths	Time	Fails	Time	Fails	
qwh-25-40	2.37	0.61	90.7	0.68	94.4	fpsol2.i.1	1.09	0.38	35.6	0.39	35.6		
qwh-25-42	2.34	0.32	33.7	0.43	44.7	inithx.i.1	1.01	0.49	24.4	0.47	24.4		
qwh-27-40	2.43	0.72	85.3	1.10	130	inithx.i.2	0	0.20	0	0.20	0		
qwh-27-42	2.39	0.58	62	1.02	117	le450_15a	2.16	3.69	2.93K	41.7	23.2K		
qwh-29-40	2.44	2.83	331	3.87	442	le450_15b	2.08	1.64	1.27K	5.27	4.2K		
qwh-29-42	2.47	3.14	360	3.06	329	le450_25a	0	0.11	0	0.11	0		
qwh-31-40	2.43	2.12	210	2.70	228	le450_25b	0	0.10	0	0.11	0		
qwh-31-42	2.40	1.14	97.9	1.62	141	le450_5a	2.04	3.06	4.42K	10.4	13.5K		
qwh-33-40	2.44	4.39	394	7.23	658	le450_5c	1.90	0.12	51	0.13	65		
qwh-33-42	2.41	2.43	200	4.12	351	miles1500	1.04	0.61	73.8	0.64	73.8		
qwh-35-40	2.44	19.2	1.51K	33.9	2.14K	queen8_8	1.90	18.1	24.1K	19.8	27.7K		
qwh-35-42	2.44	6.18	482	17.2	1.19K	queen9_9	1.93	108	79.8K	173	96.5K		

The reduction in time per choice point is roughly 20%-30% on average. Given that propagation does not make up for 100% of the work that has to be done per choice point (there are also impact updates, nogood computations, branching variable selection etc.), this reduction is substantial.

5.3 Non-dominated UIVs

Next we investigate the impact of computing non-dominated UIVs when learning nogoods. Table 2 shows the results on quasi-group with holes and graph coloring instances. We did not conduct this experiment on random binary CSPs as our solver detects quickly that most often the branching variable is the only UIV and then switches the optimization off.

Table 3. Comparison between minDomain and impact-based branching. Time in [s].

	Imp		MinDom			Imp		MinDom			Imp		MinDom	
QWH	Time	Nodes	Time	Nodes	BCSP	Time	Nodes	Time	Nodes	GraphCol	Time	Nodes	Time	Nodes
qwh-25-40	0.61	166	0.81	193	b-25-20-.43	2.04	2.91K	2.51	2.90K	fpsol2.i.1	0.38	362	0.28	375
qwh-25-42	0.32	107	0.45	132	b-25-25-.45	6.35	7.02K	7.66	6.64K	inithx.i.1	0.49	587	0.96	777
qwh-27-40	0.72	187	0.86	229	b-25-30-.47	13.9	12.6K	16.9	11.7K	inithx.i.2	0.20	558	0.22	599
qwh-27-42	0.58	179	0.83	227	b-25-40-.5	54.6	40.4K	69.2	35.5K	le450_15a	3.69	6.84K	23.2	17.6K
qwh-29-40	2.83	585	4.54	742	b-30-20-.37	18.0	24.7K	19.5	20.6K	le450_15b	1.64	3.24K	1.23	4.52K
qwh-29-42	3.14	645	2.02	551	b-30-25-.39	55.4	60.7K	62.5	51.7K	le450_25a	0.11	438	0.10	438
qwh-31-40	2.12	497	2.44	576	b-30-30-.33	1.21	1.10K	1.69	1.33K	le450_25b	0.10	438	0.09	438
qwh-31-42	1.14	350	1.38	389	b-30-40-.35	2.53	1.46K	4.89	2.09K	le450_5a	3.06	5.67K	1.62	7.86K
qwh-33-40	4.39	861	5.22	1,07K	b-35-20-.26	1.40	1.55K	1.70	1.63K	le450_5c	0.12	73.5	0.10	69.1
qwh-33-42	2.43	641	2.75	769	b-35-25-.28	5.08	4.19K	5.13	3.63K	miles1500	0.61	257	0.48	449
qwh-35-40	19.2	2.68K	12.8	2.79K	b-35-30-.29	4.85	3.19K	9.91	4.54K	queen8_8	18.1	27.8K	24.0	25.5K
qwh-35-42	6.18	1.28K	6.43	1.67K						queen9_9	108	92.1K	135	72.5K

In the table we compare two variants of our solver. The first uses the $O(mn)$ approach presented in Section 3.2 and ensures that non-dominated UIVs are used for computing the nogood. The second approach works in linear time $O(m)$ and uses the branching variable as a basis for computing the nogood.

We see clearly that using non-dominated UIVs has a profound impact on the number of failures which are almost always substantially lower than when potentially dominated nogoods are used. Interestingly, our data shows that the time per choice point is not measurably higher when using the advanced nogood learning scheme. In Table 2 we show the average number of shortest path computations for each failure. As we can see, it is usually very low, somewhere between two and three shortest paths are sufficient on average to find a non-dominated UIV.

5.4 MinDomain vs. Impacts

The work in [2] suggested that augmenting a SAT solver with min-domain branching can lead to substantial performance improvements. Since impacts have since become a popular alternative to min-domain branching in CP, we investigated which method performs better for multi-valued SAT. As Table 3 shows, on graph coloring both methods perform roughly the same, while on random binary CSPs and QWH impacts work clearly better. We also tested activity-based branching heuristics commonly used in SAT and min domain over weighted degree [6], but both were not competitive (the latter due to the large number of constraints). Our solver therefore uses impact-based branching.

5.5 MV-SAT vs. SAT

In our last experiment, we compare our multi-valued solver with the well-known MiniSAT solver for boolean SAT. In particular, we use Sugar [23] to pre-compile the SAT formulas from the XCSP model of each instance. The MV-SAT instances for our solver are generated as explained before. In our experiment, we compare the pure *solution time* of MiniSAT and CMV-SAT-1 on the resulting SAT and MV-SAT instances. Note that this solution time does not include the time that Sugar needs to compile the SAT formula, nor the time for reading in the input.

Table 4. CMV-SAT 1 vs. MiniSAT. For the QWH and BCSPs we aggregate all instances in our benchmark set that have the same domain size, for Graph Coloring and N-Queens we aggregate all instances. Time in [s], timeout for the runs is 15 min.

Class	CMV-SAT-1			MiniSAT		
	Time	Nodes	Time Outs	Time	Nodes	Time Outs
GraphCol	137	138K	0	178	373K	3
N-Queens	168	12.4K	2	235	106K	0
qwh D=25	0.93	273	0	5.35	19.6K	0
qwh D=27	1.30	366	0	13.2	36.7K	0
qwh D=29	6.00	1.23K	0	48.0	94.4K	0
qwh D=31	3.26	847	0	64.0	119K	1
qwh D=33	6.82	1.5K	0	178	218K	14
qwh D=35	25.2	4K	0	338	318K	54
QWH Total	43.5	8.26K	0	647	806K	69
b D=20	21.4	30K	0	13.8	181K	0
b D=25	66.8	72K	0	64.2	524K	1
b D=30	20.0	17K	0	22.1	165K	0
b D=40	61.1	41K	0	103	423K	0
BCSP Total	169.3	160K	0	203.1	1293K	1

Table 4 summarizes our results. We observe that CMV-SAT-1 visits massively fewer choice points than MiniSAT. Depending on the class of inputs the reduction is typically between one and two orders of magnitude. We attribute this reduction in part to impact-based branching, and in part to the use of sophisticated nogoods. Heavier inference, however, results in an increased time per choice point when compared to MiniSat. This increase is not so much due to the time needed to compute the advanced nogoods but due to the time needed to process the implications of the enhanced inference.

Overall, we find that CMV-SAT-1 performs a little better than MiniSAT on graph coloring and random BCS problems, whereby on both CMV-SAT-1 appears to work slightly more robustly and thus causes fewer timeouts. On the quasi-groups with holes problem, CMV-SAT-1 clearly outperforms MiniSAT. For all problems, the reduction in the number of choice points is very substantial and overall the multi-valued SAT solver runs up to twenty five times faster than the boolean SAT solver.

6 Conclusion

We have introduced CMV-SAT-1, a new complete multi-valued SAT solver which can serve as a back-end for CSP solvers that are based on decomposition and reformulation. We contributed the ideas of quantitative supports to augment the well-known watched literal scheme, and a new method for learning multi-valued nogoods. Experiments substantiated the practical benefits of these ideas and showed that multi-valued SAT solving offers great potential for improving classical boolean SAT technology.

References

1. Ansótegui, C.: Complete SAT solvers for Many-Valued CNF Formulas. PhD thesis, Universitat de Lleida (2004)
2. Ansótegui, C., Larrubia, J., Manyà, F.: Boosting chaff's performance by incorporating CSP heuristics. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 96–107. Springer, Heidelberg (2003)
3. Ansótegui, C., Manyà, F.: Mapping Problems with Finite-Domain Variables to Problems with Boolean Variables. In: Hoos, H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 1–15. Springer, Heidelberg (2005)
4. Ansótegui, C., Larrubia, J., Liu, C., Manyà, F.: Exploiting multivalued knowledge in variable selection heuristics for SAT solvers. *Ann. Math. Artif. Intell.* 49(1-4), 191–205 (2007)
5. Bessiere, C.: <http://www.lirmm.fr/~bessiere/generator.html>
6. Boussemart, F., Lecoutre, F., Sais, C.: Boosting systematic search by weighting constraints. In: ECAI, pp. 146–150 (2004)
7. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
8. Frisch, A., Peugniez, T.: Solving Non-Boolean Satisfiability Problems with Stochastic Local Search. In: IJCAI, pp. 282–288 (2001)
9. Graph coloring instances, <http://mat.gsia.cmu.edu/COLOR/instances.html>
10. Gent, I.: Arc consistency in SAT. In: ECAI, pp. 121–125 (2002)
11. Katsirelos, G.: Nogood Processing in CSPs. PhD Thesis, University of Toronto (2009)
12. Li, C., Anbulagan, A.: Heuristics based on unit propagation for satisfiability problems. In: IJCAI, pp. 366–371 (1997)
13. Liu, C., Kuehlmann, A., Moskewicz, M.: CAMA: A Multi-Valued Satisfiability Solver. In: ICCAD, pp. 326–333 (2003)
14. Mitchell, D.: Resolution and Constraint Satisfaction. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 554–569. Springer, Heidelberg (2003)
15. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation=Lazy Clause Generation. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 544–558. Springer, Heidelberg (2007)
16. Prosser, P.: An empirical study of phase transitions in binary constraint satisfaction problems. *Frontiers in Problem Solving: Phase Transitions and Complexity* 81(1-2), 81–109 (1996)
17. Refalo, P.: Impact-Based Search Strategies for Constraint Programming. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 557–571. Springer, Heidelberg (2004)
18. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: DAC, pp. 530–535 (2001)
19. Schiex, T., Verfaillie, G.: Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In: JAIT, pp. 48–55 (1994)
20. Selman, B., Kautz, H., Cohen, B.: Local Search Strategies for Satisfiability Testing. In: DIMACS, pp. 521–532 (1995)
21. International, C.S.P.: Competition Result Pages, <http://bach.istc.kobe-u.ac.jp/sugar/cpai08.html>
<http://bach.istc.kobe-u.ac.jp/sugar/csc09.html>
22. Tamura, N., Taga, A., Banbara, M.: System Description of a SAT-based CSP solver Sugar. In: CPAI (2008), <http://bach.istc.kobe-u.ac.jp/sugar/cpai08-sugar.pdf>
23. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling Finite Linear CSP into SAT. *Constraints* 14, 254–272 (2009)
24. Walsh, T.: SAT vs CSP. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 441–456. Springer, Heidelberg (2000)
25. Zhang, L., Madigan, C., Moskewicz, M., Malik, S.: Efficient conflict driven learning in a boolean satisfiability solver. In: ICCAD, pp. 279–285 (2001)

Exact Cover via Satisfiability: An Empirical Study

Tommi Junttila^{1,*} and Petteri Kaski²

¹ Aalto University

Department of Information and Computer Science

PO Box 15400, FI-00076 Aalto, Finland

Tommi.Junttila@tkk.fi

² Helsinki Institute for Information Technology HIIT

Aalto University

Department of Information and Computer Science

PO Box 15400, FI-00076 Aalto, Finland

Petteri.Kaski@tkk.fi

Abstract. Many tasks in combinatorial computation admit a natural formulation as instances of the exact cover problem. We observe that the exact cover problem can in turn be compactly translated to the satisfiability (SAT) problem, allowing the basic Davis-Putnam-Logemann-Loveland procedure with no clause learning to linearly simulate backtrack search for exact cover. This SAT-based approach is empirically compared with a widely applied backtrack search algorithm, Knuth's "Dancing Links X" algorithm, on a set of benchmark problems arising in combinatorial enumeration. The experimental results indicate that the current model-counting SAT solvers are in general superior in pruning the search space, but still need to be optimized for running time.

1 Introduction

An exact cover instance is a pair $P = \langle V, S \rangle$, where V is a finite set of *points* and S is a set of subsets of V . A solution to P is a subset S^* of S such that each point $v \in V$ appears in exactly one set in S^* . The exact cover problem has three natural variants: decision, counting, and enumeration. The first variant asks us to determine whether a solution exists, the second variant asks us to determine the number of solutions, and the third variant asks us to explicitly construct every solution.

The exact cover problem occupies a notable position in combinatorial mathematics, in particular in the study of highly regular structures such as *designs* (see [1]) that arise in combinatorial analysis and applications in statistics. From a computational perspective, many of the questions under study in design theory can be formulated as specific instances of the exact cover problem. For example, a celebrated result of Lam, Thiel, and Swiercz on the nonexistence of finite projective planes of order 10 [2] reduces to the observation that a specific instance of the exact cover problem on $\binom{111}{2}$ points has no solution. Besides the decision variant, also the counting and enumeration variants of the problem are of interest in combinatorics. In particular, one is recurrently interested in listing (up to isomorphism) all the structures that meet a collection of constraints,

* Financially supported by the Academy of Finland (project 122399).

where the constraints can be modeled as an instance of exact cover (see [3]). The gist of the matter is that one wants to push the envelope in terms of practical performance, even for very specific instances of exact cover.

Given the success of the SAT community in engineering for practical performance, it is warranted to put forth the question whether SAT-based techniques could be deployed in the context of exact cover. Our objective in this paper is to explore the issue.

We adopt Donald Knuth’s “Dancing Links X” (DLX) algorithm [4] as our baseline algorithm for exact cover. Algorithm DLX performs a backtrack search that covers the points in V one at a time, whereby the next point selected for covering is always a point that has the least number of choices in S for covering it. The name of the algorithm stems from the data structure that records the search state, where doubly linked lists enable fast update- and rewind operations to the state as points are covered and uncovered.

Our contributions in this paper are as follows. First, we give rather natural encodings for the exact cover problem as propositional satisfiability, pseudo-Boolean satisfiability, and stable models logic programming problems. We then compare the best-case behavior of Algorithm DLX to that of the basic, non-learning Davis-Putnam-Logemann-Loveland (DPLL) algorithm for SAT: it turns out that DPLL can simulate DLX linearly and is thus at least as efficient in theory. As a consequence, modern SAT solvers that apply also advanced search space pruning techniques such clause learning as well as restarts have the potential of solving much harder exact cover problems than DLX. To find out whether this potential realizes in practise, we evaluate the performance of an implementation of DLX against several state-of-the-art constraint solvers, especially #SAT algorithms (for a review on techniques for model counting in SAT, see [5]). As a benchmark set we use a collection of combinatorial design problems expressed as instances of exact cover. We find that the DPLL-based approaches (with and without clause learning) can indeed solve the problems by using much smaller search spaces, as suggested by the analytic comparison. Unfortunately the current implementations do not perform that well in terms of running time and some implementations run out of memory quite quickly. Our hope is that the benchmark set we provide will foster further development of constraint solvers so that in the future much harder instances of the exact cover problem could be successfully attacked.

2 SAT and DPLL Search Trees

We assume the basic definitions of Boolean variables, literals, and conjunctive normal form (CNF) formulas (e.g. [6]). A truth assignment τ for a formula ψ is a set of literals over the variables occurring in ψ ; τ is (i) *inconsistent* if both $x \in \tau$ and $\neg x \in \tau$ for some variable x , (ii) *consistent* if it is not inconsistent, and (iii) *complete* if either $x \in \tau$ or $\neg x \in \tau$ for each variable x occurring in ψ . A complete and consistent truth assignment satisfies a formula if it assigns for each clause at least one literal in the clause to true. Given a formula ψ , $\text{up}(\psi, \tau)$ denotes the set of literals implied by *unit propagation* on ψ under τ ; formally, $\text{up}(\psi, \tau)$ is the smallest set U of literals satisfying (i) $\tau \subseteq U$, and (ii) if a clause $(l_1 \vee \dots \vee l_n)$ is in ψ and $\{\neg l_1, \dots, \neg l_{i-1}, \neg l_{i+1}, \dots, \neg l_n\} \subseteq U$, then $l_i \in U$. Note that if ψ contains a unit clause (l) , then $l \in \text{up}(\psi, \tau)$ for all truth assignments τ .

Given a formula ψ , the DPLL algorithm enumerating all satisfying truth assignments of ψ can be presented as the following pseudo-code, invoked with “DPLL(ψ, \emptyset)”.

```
DPLL( $\psi, \tau$ )
  let  $\tau := \text{up}(\psi, \tau)$  /* unit propagation */
  if  $\tau$  is inconsistent or  $\psi$  contains the empty clause  $()$ , then return /* failure */
  if  $\tau$  is complete for  $\psi$ , then report the assignment  $\tau$  and return /* success */
  heuristically select a variable  $x$  in  $\psi$  but not assigned by  $\tau$ 
  call DPLL( $\psi, \tau \cup \{\neg x\}$ )
  call DPLL( $\psi, \tau \cup \{x\}$ )
```

A *DPLL search tree* of ψ is simply a call-tree of the procedure when invoked with DPLL(ψ, \emptyset). Thus each leaf node “DPLL(ψ, τ)” in a search tree is either (i) a solution node when $\text{up}(\psi, \tau)$ satisfies ψ , or (ii) a bad node when $\text{up}(\psi, \tau)$ is inconsistent.

3 SAT Encodings for Exact Cover

Assume an exact cover problem $P = \langle V, S \rangle$ and, for each point $v \in V$, denote by $S[v] = \{s \in S \mid v \in s\}$ the sets in S that include v . The problem P can be formulated as the CNF formula

$$\text{cnf}(P) := \bigwedge_{v \in V} \text{exactly-one}(\{x_s \mid s \in S[v]\}).$$

For each set $s \in S$ in the exact cover problem, the formula $\text{cnf}(P)$ has the corresponding Boolean variable x_s and the “exactly one” constraints $\text{exactly-one}(\dots)$ simply force, for each point $v \in V$, exactly one set that includes v to be in the solution. Formally, $\text{exactly-one}(X)$ over a set $X = \{x_1, \dots, x_n\}$ of Boolean variables is a CNF formula such that any complete truth assignment τ satisfies $\text{exactly-one}(X)$ if and only if exactly one variable in X is assigned to true by τ . We consider three encoding schemes to build “exactly one” constraints (see [7]). For each scheme, $\text{exactly-one}(\emptyset) := ()$ i.e. false, and $\text{exactly-one}(\{x_1\}) := (x_1)$. When $n \geq 2$, different schemes define $\text{exactly-one}(\{x_1, \dots, x_n\})$ as follows.

pairwise encoding is the most obvious scheme, simply excluding each pair in X explicitly. Thus it is composed of one n -ary clause and $(n^2 - n)/2$ binary clauses:

$$\text{exactly-one}(\{x_1, \dots, x_n\}) := (x_1 \vee \dots \vee x_n) \wedge \bigwedge_{1 \leq i < j \leq n} (\neg x_i \vee \neg x_j).$$

bitwise encoding uses $k = \lceil \log_2 n \rceil$ new auxiliary Boolean variables a_1, \dots, a_k whose values are forced to be the binary representation of i whenever x_i is true. Formally,

$$\text{exactly-one}(\{x_1, \dots, x_n\}) := (x_1 \vee \dots \vee x_n) \wedge \bigwedge_{i=1}^n \bigwedge_{j=1}^k (\neg x_i \vee l_{i,j})$$

where $l_{i,j}$ is a_j if the j th bit in the binary representation of i is 1 and $\neg a_j$ otherwise. Thus this scheme requires $n \lceil \log_2 n \rceil$ binary clauses and one n -ary clause.

ladder encoding uses $n - 1$ new auxiliary Boolean variables a_2, \dots, a_n in a way that forces an a_i to be true if and only if at least one of x_1, \dots, x_i is true. Letting $\alpha \Leftrightarrow (\beta \vee \gamma)$ abbreviate the conjunction $(\neg\alpha \vee \beta \vee \gamma) \wedge (\alpha \vee \neg\beta) \wedge (\alpha \vee \neg\gamma)$, we define

$$\text{exactly-one}(\{x_1, \dots, x_n\}) := (a_2 \Leftrightarrow (x_1 \vee x_2)) \wedge \bigwedge_{3 \leq i \leq n} (a_i \Leftrightarrow (a_{i-1} \vee x_i)) \wedge (\neg x_1 \vee \neg x_2) \wedge \bigwedge_{3 \leq i \leq n} (\neg a_{i-1} \vee \neg x_i) \wedge (a_n).$$

Thus the encoding produces one unary, $3(n - 1)$ binary, and $n - 1$ ternary clauses.

In all three different encoding schemes, given a truth assignment τ , the formula $\psi = \text{exactly-one}(X)$ has the following properties:

1. If $x_i \in \tau$ for an $x_i \in X$, then all the other variables in X are forced to false by unit propagation, i.e., $\neg x_j \in \text{up}(\psi, \tau)$ for all $x_j \in X \setminus \{x_i\}$.
2. If all but one variable in X are false, then the remaining one is forced to true by unit propagation: if $x_j \in X$ and $\neg x_i \in \tau$ for all $x_i \in X \setminus \{x_j\}$, then $x_j \in \text{up}(\psi, \tau)$.
3. If $x_i \in \tau$ for an $x_i \in X$, then the auxiliary Boolean variables used (if any) have unique values determined by unit propagation.

Based on these, it is easy to argue that the solutions of an exact cover problem $P = \langle V, S \rangle$ and its CNF encoding $\text{cnf}(P)$ have a one-to-one correspondence:

1. If $S^* \subseteq S$ is a solution of P , then there exists exactly one satisfying truth assignment τ fulfilling $\tau \supseteq \{x_s \mid s \in S^*\} \cup \{\neg x_s \mid s \notin S^*\}$ to the formula $\text{cnf}(P)$.
2. If τ is a satisfying truth assignment to the formula $\text{cnf}(P)$, then $\{s \mid s \in S \wedge x_s \in \tau\}$ is a solution of P .

3.1 Pseudo-Boolean and Logic Programming Encodings

In addition to propositional satisfiability, exact cover can also be encoded, even more naturally, as a pseudo-Boolean satisfiability problem [8] or as a stable models logic program with cardinality constraints [9]. Both of these formalisms natively support cardinality constraints and thus also the exactly-one constraint, making the encodings rather trivial.

An exact cover problem $P = \langle V, S \rangle$ can be formulated as the conjunction of linear pseudo-Boolean constraints

$$\bigwedge_{v \in V} \left(\sum_{s \in S[v]} 1 \cdot x_s = 1 \right)$$

where x_s is again the Boolean variable corresponding to the set $s \in S$ in the exact cover problem. As a logic program with cardinality constraints, the encoding is similarly just a list of fact rules so that, for each point $v \in V$ with $S[v] = \{s_1, \dots, s_k\}$, there is a cardinality constraint fact rule

$$1 \{x_{s_1}, \dots, x_{s_k}\} 1.$$

enforcing that exactly one of the atoms x_{s_1}, \dots, x_{s_k} is included in the model.

4 Algorithm DLX and Its Analytic Comparison to DPLL

Assume an exact cover problem instance $P = \langle V, S \rangle$. The Algorithm DLX for enumerating all the solutions of P can be described as the following recursive procedure.

DLX(V, S, S^*)

```

/*  $V$  are the uncovered points,  $S$  the available sets,  $S^*$  is a partial solution */
if  $V = \emptyset$ , then report the solution  $S^*$  and return /* success */
if there is an  $v \in V$  such that  $S[v] = \emptyset$ , then return /* failure */
heuristically select a point  $v \in V$  to be covered next
for each  $s \in S[v]$ , call DLX( $V \setminus s, \{s' \in S \mid s' \cap s = \emptyset\}, S^* \cup \{s\}$ )

```

A *DLX search tree* for P is the invocation tree of the procedure when initiated with the invocation DLX(V, S, \emptyset). In particular, each leaf node in the tree is either (i) a solution node of form “DLX($\emptyset, \emptyset, S^*$)” or (ii) a bad node of form “DLX(V', S', S^*)” with $S'[v] = \emptyset$ for some $v \in V'$. We observe that the standard heuristic for point selection (i.e. select a point $v \in V$ such that $|S[v]|$ is minimized, breaking ties arbitrarily) in DLX in effect induces a form of unit propagation: if in a non-leaf node “DLX(V, S, S^*)” there is a $v \in V$ such that $|S[v]| = 1$, then the heuristic guides us to augment the partial solution S^* with the unique set in $S[v]$.

When comparing DPLL and DLX, we observe that, in theory, already the basic DPLL without clause learning can linearly simulate Algorithm DLX:

Theorem 1. *If an exact cover problem $P = \langle V, S \rangle$ has a DLX search tree of size K , then the CNF formula $\text{cnf}(P)$ with any of the three encoding schemes has, subject to an idealized variable selection heuristic, a DPLL search tree of size at most $2K$.*

Proof. Sketch. Starting from the root, we simulate each k -ary branch from a DLX node “DLX(V, S, S^*)” using at most $k - 1$ binary branches on the corresponding DPLL node “DPLL($\text{cnf}(P), \tau$)” as follows. Let v be the point selected by DLX for covering. Let $S[v] = \{s_1, \dots, s_k\}$. Let $Y = \{x'_{s_1}, \dots, x'_{s_j}\} \subseteq \{x_s \mid s \in S[v]\}$ be the corresponding variables not assigned by the truth assignment τ . Then, DPLL first branches with x'_{s_1} resulting in children “DPLL($\text{cnf}(P), \text{up}(\text{cnf}(P), \tau) \cup \{x'_{s_1}\}$)” and “DPLL($\text{cnf}(P), \text{up}(\text{cnf}(P), \tau) \cup \{-x'_{s_1}\}$)”, then on the latter child with x'_{s_2} , and so on up to $x'_{s_{j-1}}$. \square

Therefore, DPLL is, when viewed as a proof system, at least as strong as DLX in solving exact cover problems. Here it is important to observe that basic DPLL corresponds to tree-like resolution, which is a strictly weaker proof system than the proof systems underlying modern SAT solvers with clause learning and restarts (see e.g. [10,11]). Thus, in theory, modern SAT solvers should allow us to solve much harder exact cover instances than DLX.

5 Experimental Results

We now compare the performance of an implementation of Algorithm DLX, namely libexact [12], to that of several constraint solvers operating on the CNF, pseudo-Boolean, and logic program encodings.

As the benchmark set (available at <http://www.tcs.hut.fi/~tjunttila/experiments/CP2010/>), we use the following families of exact cover problems of combinatorial origin; see [1] for a description of the relevant combinatorial objects.

bell-[n]. Set partitions of an n -element set. The number of solutions is the Bell number B_n .

doublefact-[2n-1]. Perfect matchings in the complete graph K_{2n} . The number of solutions is the double factorial $(2n - 1)!! = (2n)!/(2^n n!)$.

triplesys-[v]-1. Steiner triple systems of order v for small orders.

latin[n]-blk. Latin squares of order n represented as transversal designs $TD(3, n)$. One block and its neighborhood have been completed to break symmetry.

sts[v]-blk. Steiner triple systems of order v . One block and its neighborhood have been completed to break symmetry.

kts[v]-ptpt. Kirkman triple systems of order v . Two points have been completed to break symmetry. We highlight the set **kts21-ptpt** as a combinatorial challenge (the benchmark contains the first 100 instances out of a collection of 2977): a complete classification of the Kirkman triple systems of order 21 is not known, and a fast enumeration algorithm for the solutions of these 2977 instances would enable such a classification.

To highlight the role of counting and enumeration variants of the exact cover problem in connection with instances arising in combinatorics, we first compare **libexact** to the following DPLL-based SAT solvers capable of model counting and enumeration: (i) **Cachet** [13] version 1.22, (ii) **sharpSAT** [14] version 1.1b, (iii) **Satz** [15] version 215.2 modified to find all satisfying models instead of just one, (iv) **reلسat** [16] version 2.02, and (v) **clasp** [17] version 1.3.2. The **#2clseq** [18] and **CQuest** [19] systems were not tested as they seem not to be publicly available. Figure 1 shows representative digests of the results. The search spaces of the DPLL-based approaches are generally smaller than those of **libexact**; as the best example, Fig. 1(a) shows that **sharpSAT** has one order of magnitude smaller search space than **libexact** in most cases and seems to scale at least polynomially better on the **doublefact** family. It should be noted that the search spaces of some advanced DPLL-based model counting approaches, e.g. **Cachet** and **sharpSAT**, can be smaller than the number of solutions due to the dynamic problem decomposition and caching techniques applied. Unfortunately, when we compare running times, Fig. 1(b) for **sharpSAT** and Fig. 1(c) for **clasp**, we see that on nontrivial benchmarks the DPLL-based approaches are consistently one to two orders of magnitude slower than **libexact** (with the exception of **sharpSAT** on the **doublefact** family). In addition, **Cachet** and **sharpSAT** run out of 4GB of memory on some instances. Of the DPLL-based approaches tested, **clasp** has the best run time behavior and is also very insensitive to the applied “exactly one” encoding scheme. For other solvers, the bitwise and pairs encoding schemes were slightly better than the ladder scheme but the pairs encoding sometimes results in prohibitively large CNF formulas.

An approach to #SAT that is not based on DPLL is taken in **c2d** [20]: the CNF formula is translated into a deterministic and decomposable normal form from which it is easy to compute the number of satisfying truth assignments. The run time results on all families with the ladder encoding are shown in Fig. 1(d). This method scales better than **libexact** on the **doublefact** family having a quite small point set and a very

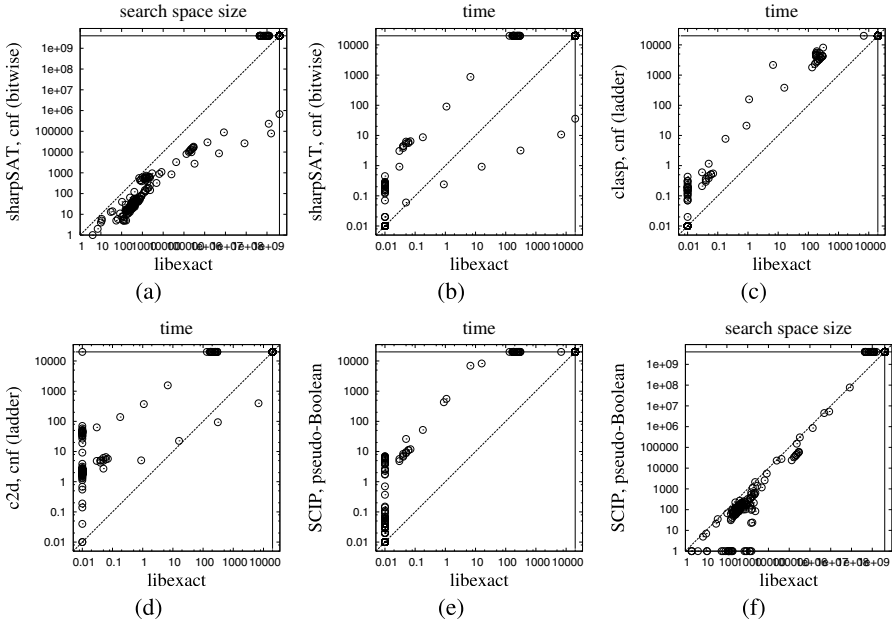


Fig. 1. Some summary results comparing `libexact` to constraint solver approaches

large number of solutions; interestingly, this does only happen with the ladder encoding involving the largest number of variables of all three CNF encodings presented, on the other encodings the run times are much worse than those of `libexact`. On other families, the method is not competitive.

We also experiment with the publicly available, state-of-the-art pseudo-Boolean solver SCIP [21] version 1.2.0 (linked with SoPlex version 1.4.2) that is based on combining constraint and mixed integer programming, and also capable of counting solutions [22]. As in the DPLL-case, the results in Figs. 1(e) and 1(f) show that the search spaces are generally smaller but the run times much worse than those of `libexact`.

For the stable models logic program encoding, we again apply the stable models solver `clasp` [17]. Even though `clasp` does not perform CNF translation internally, the results are essentially the same as obtained when running `clasp` as a #SAT solver on the CNF encodings, further highlighting the robustness of `clasp` on these benchmarks.

Finally, we would like to stress that the `kts21-ptpt` family presents a real challenge: none of the approaches solved any of the tested instances. In fact, the state-of-the-art SAT (decision) solver `minisat2` [23] version 070721 was not able to solve the satisfiability of any of the 20 instances tested within four hours. On the other benchmark families `minisat2` was able to deduce the satisfiability of each instance in less than one second (except on the larger `bell` instances with bitwise and pairs encodings, where the formula simplifying preprocessor consumes a lot of time if switched on).

In conclusion, we observe that instances of exact cover translated into satisfiability present challenging benchmarks for the development of practical #SAT solvers. Furthermore, we observe that solvers incorporating clause learning with caching and

dynamic problem decomposition exhibit superior performance in terms of search space reduction; if it is possible to integrate these techniques with the fast and memory-conservative algorithms of clasp, Algorithm DLX could perhaps be beaten also in terms of run time.

References

1. Colbourn, C.J., Dinitz, J.H.: Handbook of Combinatorial Designs, 2nd edn. Chapman & Hall/CRC (2007)
2. Lam, C.W.H., Thiel, L., Swiercz, S.: The nonexistence of finite projective planes of order 10. *Canadian Journal of Mathematics* 41(6), 1117–1123 (1989)
3. Kaski, P., Östergård, P.R.J.: Classification Algorithms for Codes and Designs. Springer, Heidelberg (2006)
4. Knuth, D.E.: Dancing links. arXiv:cs/0011047 (November 2000)
5. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting. In: [6], pp. 633–654
6. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press, Amsterdam (2009)
7. Prestwich, S.: CNF encodings. In: [6], pp. 75–97
8. Roussel, O., Manquinho, V.M.: Pseudo-boolean and cardinality constraints. In: [6], pp.695–733
9. Simons, P., Niemelä, I., Soinen, T.: Extending and implementing the stable model semantics. *Artificial Intelligence* 138(1-2), 181–234 (2002)
10. Beame, P., Kautz, H.A., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research* 22, 319–351 (2004)
11. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers with restarts. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 654–668. Springer, Heidelberg (2009)
12. Kaski, P., Pottonen, O.: libexact user’s guide. HIIT Technical Reports 2008-1, Helsinki Institute for Information Technology HIIT (2008)
13. Sang, T., Beame, P., Kautz, H.A.: Heuristics for fast exact model counting. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 226–240. Springer, Heidelberg (2005)
14. Thurley, M.: sharpSAT — counting models with advanced component caching and implicit BCP. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 424–429. Springer, Heidelberg (2006)
15. Li, C.M., Anbulagan: Look-ahead versus look-back for satisfiability problems. In: Smolka, G. (ed.) CP 1997. LNCS, vol. 1330, pp. 341–355. Springer, Heidelberg (1997)
16. Bayardo Jr., R.J., Pehoushek, J.D.: Counting models using connected components. In: Proc. AAAI/IAAI 2000, pp. 157–162. AAAI Press/The MIT Press (2000)
17. Gebser, M., Kaufmann, B., Schaub, T.: Solution enumeration for projected boolean search problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 71–86. Springer, Heidelberg (2009)
18. Davies, J., Bacchus, F.: Using more reasoning to improve #SAT solving. In: Proc. AAAI 2007, pp. 185–190. AAAI Press, Menlo Park (2007)
19. Morgado, A., Marques-Silva, J.: Good learning and implicit model enumeration. In: Proc. ICTAI 2005, pp. 131–136. IEEE Computer Society, Los Alamitos (2005)
20. Darwiche, A.: New advances in compiling CNF to decomposable negation normal form. In: Proc. ECAI 2004, pp. 328–332. IOS Press, Amsterdam (2004)
21. Achterberg, T.: Constraint Integer Programming. PhD thesis, Technische Universität Berlin (2007), <http://opus.kobv.de/tuberlin/volltexte/2007/1611/>
22. Achterberg, T., Heinz, S., Koch, T.: Counting solutions of integer programs using unrestricted subtree detection. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 278–282. Springer, Heidelberg (2008)
23. Eén, N., Sörensson, N.: An extensible SAT solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)

On the Complexity and Completeness of Static Constraints for Breaking Row and Column Symmetry*

George Katsirelos¹, Nina Narodytska², and Toby Walsh²

¹ CRIL-CNRS, Lens, France
gkatsi@gmail.com

² NICTA and University of NSW, Sydney, Australia
{nina.narodytska,toby.walsh}@nicta.com.au

Abstract. We consider a common type of symmetry where we have a matrix of decision variables with interchangeable rows and columns. A simple and efficient method to deal with such row and column symmetry is to post symmetry breaking constraints like `DOUBLELEX` and `SNAKELEX`. We provide a number of positive and negative results on posting such symmetry breaking constraints. On the positive side, we prove that we can compute in polynomial time a unique representative of an equivalence class in a matrix model with row and column symmetry if the number of rows (or of columns) is bounded and in a number of other special cases. On the negative side, we show that whilst `DOUBLELEX` and `SNAKELEX` are often effective in practice, they can leave a large number of symmetric solutions in the worst case. In addition, we prove that propagating `DOUBLELEX` completely is NP-hard. Finally we consider how to break row, column and value symmetry, correcting a result in the literature about the safeness of combining different symmetry breaking constraints. We end with the first experimental study on how much symmetry is left by `DOUBLELEX` and `SNAKELEX` on some benchmark problems.

1 Introduction

One challenge in constraint programming is to develop effective search methods to deal with common modelling patterns. One such pattern is row and column symmetry [1]: many problems can be modelled by a matrix of decision variables [2] where the rows and columns of the matrix are fully or partially interchangeable. Such symmetry is a source of combinatorial complexity. It is therefore important to develop techniques to deal with this type of symmetry. We study here simple constraints that can be posted to break row and column symmetries, and analyse their effectiveness both theoretically and experimentally. We prove that we can compute in polynomial time the lexicographically smallest representative of an equivalence class in a matrix model with row and column symmetry if the number of rows (or of columns) is bounded and thus remove all symmetric solutions. We are therefore able for the first time to see how much symmetry is left by these commonly used symmetry breaking constraints.

* Supported by ANR UNLOC project, ANR 08-BLAN-0289-01 and the Australian Government's Department of Broadband, Communications and the Digital Economy and the ARC.

2 Formal Background

A constraint satisfaction problem (CSP) consists of a set of variables, each with a domain of values, and a set of constraints specifying allowed values for subsets of variables. When solving a CSP, we often use propagation algorithms to prune the search space by enforcing properties like domain consistency. A constraint is *domain consistent (DC)* iff when a variable in the scope of a constraint is assigned any value in its domain, there exist compatible values in the domains of all the other variables in the scope of the constraint. A CSP is domain consistent iff every constraint is domain consistent. An important feature of many CSPs is symmetry. Symmetries can act on variables or values (or both). A *variable symmetry* is a bijection σ on the variable indices that preserves solutions. That is, if $\{X_i = a_i \mid i \in [1, n]\}$ is a solution then $\{X_{\sigma(i)} = a_i \mid i \in [1, n]\}$ is also. A *value symmetry* is a bijection θ on the values that preserves solutions. That is, if $\{X_i = a_i \mid i \in [1, n]\}$ is a solution then $\{X_i = \theta(a_i) \mid i \in [1, n]\}$ is also. A simple but effective method to deal with symmetry is to add *symmetry breaking constraints* which eliminate symmetric solutions. For example, Crawford *et al.* proposed the general lex-leader method that posts lexicographical ordering constraints to eliminate all but the lexicographically least solution in each symmetry class [3]. Many problems are naturally modelled by a matrix of decision variables with variable symmetry in which the rows and/or columns are interchangeable [1]. We say that a CSP containing a matrix of decision variables has row symmetry iff given a solution, any permutation of the rows is also a solution. Similarly, it has column symmetry iff given a solution, any permutation of the columns is also a solution.

Running example: *The Equidistant Frequency Permutation Array (EFPA) problem is a challenging problem in coding theory. The goal is to find a set of v code words, each of length $q\lambda$ such that each word contains λ copies of the symbols 1 to q , and each pair of code words is Hamming distance d apart. For example, for $v = 4$, $\lambda = 2$, $q = 3$, $d = 4$, one solution is:*

$$\begin{array}{cccccc} 0 & 2 & 1 & 2 & 0 & 1 \\ 0 & 2 & 2 & 1 & 1 & 0 \\ 0 & 1 & 0 & 2 & 1 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \end{array} \tag{a}$$

This problem has applications in communication theory, and is related to other combinatorial problems like finding orthogonal Latin squares. Huczynska et al. [4] consider a model for this problem with a v by $q\lambda$ array of variables with domains 1 to q . This model has row and column symmetry since we can permute the rows and columns and still have a solution.

3 Breaking Row and Column Symmetry

To break all row symmetry we can post lexicographical ordering constraints on the rows. Similarly, to break all column symmetry we can post lexicographical ordering constraints on the columns. When we have both row and column symmetry, we can post a DOUBLELEX constraint that lexicographically orders both the rows and columns

[1]. This does not eliminate all symmetry since it may not break symmetries which permute both rows and columns. Nevertheless, it is often effective in practice.

Running example: Consider again solution (a). If we order the rows of (a) lexicographically, we get a solution with lexicographically ordered rows and columns:

$$\begin{array}{rcl}
 0\ 2\ 1\ 2\ 0\ 1 & & 0\ 0\ 1\ 1\ 2\ 2 \\
 0\ 2\ 2\ 1\ 1\ 0 & \text{order} & 0\ 1\ 0\ 2\ 1\ 2 \\
 0\ 1\ 0\ 2\ 1\ 2 & \Rightarrow & 0\ 2\ 1\ 2\ 0\ 1 \\
 0\ 0\ 1\ 1\ 2\ 2 & \text{rows} & 0\ 2\ 2\ 1\ 1\ 0
 \end{array} \tag{b}$$

Similarly if we order the columns of (a) lexicographically, we get a different solution in which both rows and columns are again ordered lexicographically:

$$\begin{array}{rcl}
 0\ 2\ 1\ 2\ 0\ 1 & & 0\ 0\ 1\ 1\ 2\ 2 \\
 0\ 2\ 2\ 1\ 1\ 0 & \text{order} & 0\ 1\ 0\ 2\ 1\ 2 \\
 0\ 1\ 0\ 2\ 1\ 2 & \Rightarrow & 0\ 1\ 2\ 0\ 2\ 1 \\
 0\ 0\ 1\ 1\ 2\ 2 & \text{cols} & 0\ 2\ 2\ 1\ 1\ 0
 \end{array} \tag{c}$$

All three solutions are thus in the same row and column symmetry class. However, both (b) and (c) satisfy the DOUBLELEX constraint. Therefore DOUBLELEX can leave multiple solutions in each symmetry class.

The lex-leader method breaks all symmetry by ensuring that any solution is the lexicographically smallest in its symmetry class [3]. This requires linearly ordering the matrix. Lexicographically ordering the rows and columns is consistent with a linearization that takes the matrix in row-wise order (i.e. appending rows in order). We therefore consider a complete symmetry breaking constraint ROWWISELEXLEADER which ensures that the row-wise linearization of the matrix is lexicographically smaller than all its row or column permutations, or compositions of row and column permutations.

Running example: Consider the symmetric solutions (a) to (c). If we linearize these solutions row-wise, the first two are lexicographically larger than the third. Hence, the first two solutions are eliminated by the ROWWISELEXLEADER constraint.

ROWWISELEXLEADER breaks all row and column symmetries. Unfortunately, posting such a constraint is problematic since it is NP-hard to check if a complete assignment satisfies ROWWISELEXLEADER [5,6]. We now give our first major result. We prove that if we can bound the number of rows (or columns), then there is a polynomial time method to break all row and column symmetry. For example, in the EFPA problem, the number of columns might equal the fixed word size of our computer.

Theorem 1. For a n by m matrix, we can check if a complete assignment satisfies a ROWWISELEXLEADER constraint in $O(n!nm \log m)$ time.

Proof: Consider the matrix model $X_{i,j}$. We exploit the fact that with no row symmetry and just column symmetry, lexicographically ordering the columns gives the lex-leader assignment. Let $Y_{i,j} = X_{\sigma(i),j}$ be a row permutation of $X_{i,j}$. To obtain $Z_{i,j}$, the smallest column permutation of $Y_{i,j}$ we lexicographically sort the m columns of $Y_{i,j}$ in $O(nm \log(m))$ time. Finally, we check that $[X_{1,1}, \dots, X_{1,m}, \dots, X_{n,1}, \dots, X_{n,m}] \leq_{\text{lex}} [Z_{1,1}, \dots, Z_{1,m}, \dots, Z_{n,1}, \dots, Z_{n,m}]$, where \leq_{lex} is the lexicographic comparison of

two vectors. This ensures that $X_{i,j}$ is lexicographically smaller than or equal to any column permutation of this row permutation. If we do this for each of the $n! - 1$ non-identity row permutations, then $X_{i,j}$ is lexicographically smaller than or equal to any row permutation. This means that we have the lex-leader assignment. This can be done in time $O(n!nm \log m)$, which for bounded n is polynomial. \square

This result easily generalizes to when rows and columns are partially interchangeable. In the experimental section, we show that this gives an effective method to break *all* row and column symmetry.

4 Double Lex

When the number of both rows and columns is large, breaking all row and column symmetry is computationally challenging. In this situation, we can post a DOUBLELEX constraint [11]. However, as we saw in the running example, this may not break all symmetry. In fact, it can leave $n!$ symmetric solutions in an $2n \times 2n$ matrix model.

Theorem 2. *There exists a class of $2n$ by $2n$ 0/1 matrix models on which DOUBLELEX leaves $n!$ symmetric solutions, for all $n \geq 2$.*

Proof: Consider a $2n$ by $2n$ matrix model with the constraints that the matrix contains $3n$ non-zero entries, and each row and column contains between one and two non-zero entries. This model has row and column symmetry since row and column permutations leave the constraints unchanged. There exists a class of symmetric solutions to the problem that satisfy a DOUBLELEX constraint of the form:

$$\begin{matrix} 0 & I^R \\ I^R & P \end{matrix}$$

Where 0 is a n by n matrix of zeroes, I^R is the reflection of the identity matrix, and P is any permutation matrix (a matrix with one non-zero entry on each row and column). For example, as there are exactly two possible permutation matrices of order 2, there are two symmetric 4 by 4 solutions with lexicographically ordered rows and columns:

$$\begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{matrix} \quad \text{and} \quad \begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{matrix}$$

In general, there are $n!$ row and column symmetries of P . Hence, DOUBLELEX leaves $n!$ symmetric solutions. \square

Having decided to break row and column symmetry with DOUBLELEX, how do we propagate it? One option is to decompose it into two LEXCHAIN constraints, one on the rows and the other on the columns. A LEXCHAIN constraint ensures that a sequence of vectors are lexicographically ordered. Enforcing domain consistency on each

LEXCHAIN constraint takes polynomial time [7]. However, this decomposition hinders propagation. For example, in the matrix of decision variables with domains:

$$\begin{matrix} 0/1 & 0/1 & 1 \\ 0/1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$$

LEXCHAIN constraints on the rows and columns ensure the second row is lexicographically larger than the first row and lexicographically smaller than the third, and the second column is lexicographically larger than the first column and lexicographically smaller than the third. Both such LEXCHAIN constraints are DC. However, the corresponding DOUBLELEX constraint is not since there is no solution in which the top left variable is set to 1. We might therefore consider a specialized propagator for the DOUBLELEX constraint. Unfortunately, whilst checking a DOUBLELEX constraint takes polynomial time, enforcing DC on this constraint is NP-hard. Thus, even when posting just DOUBLELEX to break row and column symmetry, there are computational limits on our ability to prune symmetric branches from the search tree.

Theorem 3. *Enforcing DC on the DOUBLELEX constraint is NP-hard.*

Proof: (Outline) We reduce an instance of 1-in-3SAT on positive clauses to a partially instantiated instance of the DOUBLELEX constraint with 0/1 variables. The constructed DOUBLELEX constraint has a solution iff the 1-in-3 SAT formula is satisfiable. Hence, it is NP-hard to enforce DC on the DOUBLELEX constraint [5], even with a bounded number of values. The full proof appears in [8]. □

5 Special Cases

We consider two special cases where we can check a constraint that breaks all row and column symmetry in polynomial time. In both cases, we show that we can do even better than check the constraint in polynomial time. We prove that in these cases we can enforce DC on a constraint that breaks all row and column symmetry in polynomial time. This provides a counterpoint to our result that enforcing DC on DOUBLELEX is NP-hard in general.

5.1 All-Different Matrices

An all-different matrix is a matrix model in which every value is different. It was shown in [1] that when an all-different matrix has row and column symmetry, then ROWWISELEXLEADER is equivalent to ensuring that the top left entry is the smallest value, and the first row and column are ordered. Let ORDER1STROWCOL be such a symmetry breaking constraint.

Theorem 4. *DC can be enforced on ORDER1STROWCOL in polynomial time.*

Proof: Consider the n by m matrix model $X_{i,j}$. We post $O(nm)$ constraints: $X_{1,1} < \dots < X_{n,1}, X_{1,1} < \dots < X_{1,m}, X_{1,1} < X_{1+i,1+j}$ for $1 \leq i < n$ and $1 \leq j < m$.

The constraint graph of this decomposition is acyclic. Therefore enforcing DC on the decomposition achieves DC on ORDER1STROWCOL. Each constraint in the decomposition can be made DC in constant time (assuming we can change bounds in constant time). Hence, DC can be enforced on ORDER1STROWCOL in $O(nm)$ time. \square

Note that, when applied to an all-different matrix with row and column symmetry, the general method for breaking symmetry in all-different problems proposed in [9] will post binary inequalities logically equivalent to ORDER1STROWCOL.

5.2 Matrix Models of Functions

A matrix model of a function is one in which all entries are 0/1 and each row sum is 1. If a matrix model of a function has row and column symmetry then ROWWISELEXLEADER ensures the rows and columns are lexicographically ordered, the row sums are 1, and the sums of the columns are in decreasing order, as was shown in [10,11]. We denote this symmetry breaking constraint as DOUBLELEXCOLSUM. Enforcing DC on DOUBLELEXCOLSUM takes polynomial time, in contrast to partial row and column interchangeability in matrix models of functions, which is NP-hard [12].

Theorem 5. *DC can be enforced on DOUBLELEXCOLSUM in polynomial time.*

Proof: We will show that DOUBLELEXCOLSUM can be encoded with a set of REGULAR constraints. Consider the n by m matrix model $X_{i,j}$. For each row i we introduce an extra variable Y_i and a REGULAR constraint on $[X_{i,1}, \dots, X_{i,m}, \#, Y_i]$ where $\#$ is a delimiter between $X_{i,m}$ and Y_i . Each REGULAR constraint ensures that exactly one position in the i th row is set to 1 and the variable Y_i stores this position. The automaton's states are represented by the 3-tuple $\langle s, d, p \rangle$ where s is the row sum, d is the current position and p records the position of the 1 on this row. This automaton has $4m$ states and a constant number of transitions from each state, so the total number of transitions is $O(m)$. The complexity of propagating this constraint is $O(m^2)$. We also post a REGULAR constraint over Y_1, \dots, Y_n to ensure that they form a decreasing sequence of numbers and the number of occurrences of each value is decreasing. The first condition ensures that rows and columns are lexicographically ordered and the second condition ensures that the sums of the columns are decreasing. The states of this automaton are 3-tuples $\langle v, s, r \rangle$ where v is the last value, s is the number of occurrences of this value, and r is the number of occurrences of the previous value. This automaton has $O(n^2m)$ states, while the number of transition from each state is bounded. Therefore propagating this constraint requires time $O(n^3m)$. This decomposition is logically equivalent to the DOUBLELEXCOLSUM constraint, therefore it is sound. Completeness follows from the fact that the decomposition has a Berge acyclic constraint graph. Therefore, enforcing DC on each REGULAR constraint enforces DC on DOUBLELEXCOLSUM in $O(m^2n + n^3m)$ time. \square

6 Value Symmetry

Problems with row and column symmetry also often contain value symmetries. For example, the EFPA problem has row, column and value symmetry. We therefore turn to the problem of breaking row, column and value symmetry.

Running example: Consider again the solution (a). If we interchange the values 1 and 2, we get a symmetric solution:

$$\begin{array}{ccc}
 0\ 2\ 1\ 2\ 0\ 1 & & 0\ 1\ 2\ 1\ 0\ 2 \\
 0\ 2\ 2\ 1\ 1\ 0 & \Rightarrow & 0\ 1\ 1\ 2\ 2\ 0 \\
 0\ 1\ 0\ 2\ 1\ 2 & (1\ 2) & 0\ 2\ 0\ 1\ 2\ 1 \\
 0\ 0\ 1\ 1\ 2\ 2 & & 0\ 0\ 2\ 2\ 1\ 1
 \end{array} \tag{d}$$

In fact, all values in this CSP are interchangeable.

How do we break value symmetry in addition to breaking row and column symmetry? For example, Huczynska *et al.* write about their first model of the EFPA problem:

“To break some of the symmetry, we apply lexicographic ordering (lex-ordering) constraints to the rows and columns . . . These two constraint sets do not explicitly order the symbols. It would be possible to order the symbols by using value symmetry breaking constraints. However we leave this for future work.” (page 53 of [4])

We turn to this future work of breaking row, column and value symmetry.

6.1 Double Lex

We first note that the interaction of the problem and DOUBLELEX constraints can in some circumstances break all value symmetry. For instance, in our (and Huczynska *et al.*’s) model of the EFPA problem, all value symmetry is already eliminated. This appears to have been missed by [4].

Running example: Consider any solution of the EFPA problem which satisfies DOUBLELEX (e.g. (b) or (c)). By ordering columns lexicographically, DOUBLELEX ensures that the first row is ordered. In addition, the problem constraints ensure λ copies of the symbols 1 to q to appear in the first row. Hence, the first row is forced to be:

$$\underbrace{1 \dots 1}_{\lambda} \underbrace{2 \dots 2}_{\lambda} \dots \underbrace{q \dots q}_{\lambda}$$

All value symmetry is broken as we cannot permute the occurrences of any of the values.

6.2 Puget’s Method

In general, value symmetries may remain after we have broken row and column symmetry. How can we eliminate these value symmetries? Puget has given a general method for breaking any number of value symmetries in polynomial time [13]. Given a surjection problem in which all values occur at least once, he introduces variables Z_j to represent the index of the first occurrence of each value:

$$\begin{aligned}
 X_i = j &\Rightarrow Z_j \leq i \\
 Z_j = i &\Rightarrow X_i = j
 \end{aligned}$$

¹ Any problem can be turned into a surjection problem by the addition of suitable new variables.

Value symmetry on the X_i is transformed into variable symmetry on the Z_j . This variable symmetry is especially easy to break as the Z_j take all different values. We simply need to post appropriate ordering constraints on the Z_j . Consider, for example, the inversion symmetry which maps 1 onto m , 2 onto $m - 1$, etc. Puget's method breaks this symmetry with the single ordering constraint: $Z_1 < Z_m$. Unfortunately Puget's method for breaking value symmetry is not compatible in general with breaking row and column symmetry using ROWWISELEXLEADER. This corrects Theorem 6 and Corollary 7 in [13] which claim that, provided we use the same ordering of variables in each method, it is compatible to post lex-leader constraints to break variable symmetry and Puget's constraints to break value symmetry. There is no ordering of variables in Puget's method which is compatible with breaking row and column symmetry using the lex-leader method (or any method like DOUBLELEX based on it).

Theorem 6. *There exist problems on which posting ROWWISELEXLEADER and applying Puget's method for breaking value symmetry remove all solutions in a symmetry class irrespective of the ordering on variables used by Puget's method.*

Proof: Consider a 3 by 3 matrix model with constraints that all values between 0 and 8 occur, and that the average of the non-zero values along every row and column are all different from each other. This problem has row and column symmetry since we can permute any pair of rows or columns without changing the average of the non-zero values. In addition, it has a value symmetry that maps i onto $9 - i$ for $i > 0$. This maps an average of a onto $9 - a$. If the averages were all-different before they remain so after. Consider the following two solutions:

$$\begin{array}{ccc} 0 & 2 & 3 \\ 4 & 8 & 5 \\ 7 & 6 & 1 \end{array} \quad \text{and} \quad \begin{array}{ccc} 0 & 2 & 3 \\ 4 & 1 & 5 \\ 7 & 6 & 8 \end{array}$$

Both matrices satisfy ROWWISELEXLEADER as the smallest entry occurs in the top left corner and both the first row and column are ordered. They are therefore both the lex leader members of their symmetry class.

Puget's method for breaking value symmetry will simply ensure that the first occurrence of 1 in some ordering of the matrix is before that of 8 in the same ordering. However, comparing the two solutions, it cannot be the case that the middle square is both before *and* after the bottom right square in the given ordering used by Puget's method. Hence, whichever ordering of variables is used by Puget's method, one of these solutions will be eliminated. All solutions in this symmetry class are thus eliminated. \square

We can pinpoint the mistake in Puget's proof which allows him to conclude incorrectly that his method for value symmetry can be safely combined with variable symmetry breaking methods like DOUBLELEX. Puget introduces a matrix of 0/1 variables $Y_{ij} \iff X_i = j$ and observes that variable symmetries σ on variables X_i correspond to row symmetries on the matrix Y_{ij} , while value symmetries θ of the variables X_i correspond to column symmetries of the matrix. Using the lex-leader method on a column-wise linearisation of the matrix, he derives the value symmetry breaking constraints on the Z variables. Finally, he claims that we can derive the variable symmetry breaking constraints on the X variables with the same method (equation (13) of [13]).

However, this requires a row-wise linearisation of the matrix. Unfortunately, combining symmetry breaking constraints based on row and column-wise linearisations can, as in our example, eliminate all solutions in a symmetry class.

In fact, we can give an even stronger counter-example to Theorem 6 in [13] which shows that it is incompatible to post together variable and value symmetry breaking constraints *irrespective* of the orderings of variables used by *both* the variable and the value symmetry breaking method.

Theorem 7. *There exist problems on which posting lex-leader constraints to break variable symmetries and applying Puget's method to break value symmetries remove all solutions in a symmetry class irrespective of the orderings on variables used by both methods.*

Proof: Consider variables X_1 to X_4 taking values 1 to 4, an all-different constraint over X_1 to X_4 and a constraint that the neighbouring differences are either all equal or are not an arithmetic sequence. These constraints permit solutions like $X_1, \dots, X_4 = 1, 2, 3, 4$ (neighbouring differences are all equal) and $X_1, \dots, X_4 = 2, 1, 4, 3$ (neighbouring differences are not an arithmetic sequence). They rule out assignments like $X_1, \dots, X_4 = 3, 2, 4, 1$ (neighbouring differences form the arithmetic sequence 1, 2, 3). This problem has a variable symmetry σ which reflects a solution, swapping X_1 with X_4 , and X_2 with X_3 , and a value symmetry θ that inverts a solution, swapping 1 with 4, and 2 with 3. Consider $X_1, \dots, X_4 = 2, 4, 1, 3$ and $X_1, \dots, X_4 = 3, 1, 4, 2$. These two assignments form a symmetry class of solutions.

Suppose we break variable symmetry with a lex-leader constraint on X_1 to X_4 . This will permit the solution $X_1, \dots, X_4 = 2, 4, 1, 3$ and eliminate the solution $X_1, \dots, X_4 = 3, 1, 4, 2$. Suppose we break the value symmetry using Puget's method on the same ordering of variables. This will ensure that 1 first occurs before 4. But this will eliminate the solution $X_1, \dots, X_4 = 2, 4, 1, 3$. Hence, all solutions in this symmetry class are eliminated. In this case, both variable and value symmetry breaking use the same order on variables. However, we can show that all solutions in at least one symmetry class are eliminated whatever the orders used by both the variable and value symmetry breaking.

The proof is by case analysis. In each case, we consider a set of symmetry classes of solutions, and show that the combination of the lex-leader constraints to break variable symmetries and Puget's method to break value symmetries eliminates all solutions from one symmetry class. In the first case, suppose the variable and value symmetry breaking constraints eliminate $X_1, \dots, X_4 = 3, 1, 4, 2$ and permit $X_1, \dots, X_4 = 2, 4, 1, 3$. In the second case, suppose they eliminate $X_1, \dots, X_4 = 2, 4, 1, 3$ and permit $X_1, \dots, X_4 = 3, 1, 4, 2$. This case is symmetric to the first except we need to reverse the names of the variables throughout the proof. We therefore consider just the first case. In this case, the lex-leader constraint breaks the variable symmetry by putting either X_1 first in its ordering variables or X_3 first.

Suppose X_1 goes first in the ordering used by the lex-leader constraint. Puget's method ensures that the first occurrence of 1 is before that of 4. Puget's method therefore uses an ordering on variables which puts X_3 before X_2 . Consider now the symmetry class of solutions: $X_1, \dots, X_4 = 2, 1, 4, 3$ and $X_1, \dots, X_4 = 3, 4, 1, 2$. Puget's method eliminates the first solution as 4 occurs before 1 in any ordering that put X_3 before X_2 .

And the lex-leader constraint eliminates the second solution as X_1 is larger than its symmetry X_4 . Therefore all solutions in this symmetry class are eliminated.

Suppose, on the other hand, X_3 goes first in the lex-leader constraint. Consider now the symmetry class of solutions: $X_1, \dots, X_4 = 1, 2, 3, 4$ and $X_1, \dots, X_4 = 4, 3, 2, 1$. The lex-leader constraint eliminates the first solution as X_3 is greater than its symmetry X_2 . Suppose now that the second solution is not eliminated. Puget’s method ensures the first occurrence of 1 is before that of 4. Puget’s method therefore uses an ordering on variables which puts X_4 before X_1 . Consider now the symmetry class of solutions: $X_1, \dots, X_4 = 1, 3, 2, 4$ and $X_1, \dots, X_4 = 4, 2, 3, 1$. Puget’s method eliminates the first solution as 4 occurs before 1 in any ordering that put X_4 before X_1 . And the lex-leader constraint eliminates the second solution as X_3 is larger than its symmetry X_2 . Therefore all solutions in this symmetry class are eliminated. \square

6.3 Value Precedence

We end with a special but common case where variable and value symmetry breaking do not conflict. When values partition into interchangeable sets, Puget’s method is equivalent to breaking symmetry by enforcing value precedence [14][15]. Given any two interchangeable values i and j with $i < j$, a value PRECEDENCE constraint ensures that if i occurs then the first occurrence of i is before that of j . It is safe to break row and column symmetry with ROWWISELEXLEADER and value symmetry with PRECEDENCE when value precedence considers variables either in a row-wise or in a column-wise order. This is a simple consequence of Theorem 1 in [14]. It follows that it is also safe to use PRECEDENCE to break value symmetry when using constraints like DOUBLELEX derivable from the lex-leader method.

7 Snake Lex

A promising alternative to DOUBLELEX for breaking row and column symmetries is SNAKELEX [16]. This is also derived from the lex leader method, but now applied to a snake-wise unfolding of the matrix. To break column symmetry, SNAKELEX ensures that the first column is lexicographically smaller than or equal to both the second and third columns, the reverse of the second column is lexicographically smaller than or equal to the reverse of both the third and fourth columns, and so on up till the penultimate column is compared to the final column. To break row symmetry, SNAKELEX ensures that each neighbouring pair of rows, $X_{1,i}, \dots, X_{n,i}$ and $X_{1,i+1}, \dots, X_{n,i+1}$ satisfy the entwined lexicographical ordering:

$$\langle X_{1,i}, X_{2,i+1}, X_{3,i}, X_{4,i+1}, \dots \rangle \leq_{\text{lex}} \langle X_{1,i+1}, X_{2,i}, X_{3,i+1}, X_{4,i}, \dots \rangle$$

Like DOUBLELEX, SNAKELEX is an incomplete symmetry breaking method. In fact, like DOUBLELEX, it may leave a large number of symmetric solutions.

Theorem 8. *There exists a class of $2n$ by $2n+1$ 0/1 matrix models on which SNAKELEX leaves $O(4^n / \sqrt{n})$ symmetric solutions, for all $n \geq 2$.*

Proof: Consider the following 4 by 4 matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

This is a permutation matrix as there is a single 1 on each row and column. It satisfies the SNAKELEX constraints. In fact, we can add any 5th column which reading top to bottom is lexicographically larger than or equal to 0010 and reading bottom to top is lexicographically larger than or equal to 0010. We shall add a 4 bit column with 2 bits set. That is, reading top to bottom: 1100, 1010, 0110 or 0011. Note that all 4 of these 4 by 5 matrices are row and column symmetries of each other. For instance, consider the row and column symmetry σ that reflects the matrix in the horizontal axis, and swaps the 1st column with the 2nd, and the 3rd with the 4th:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \\ \Leftrightarrow \\ \sigma \\ \end{matrix} \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

In general, we consider the $2n$ by $2n$ permutation matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{pmatrix}$$

This satisfies the SNAKELEX constraints. We can add any $2n + 1$ th column which reading top to bottom is lexicographically larger than or equal to the $2n - 1$ th column and reading bottom to top is lexicographically larger than or equal to the $2n$ th column. In fact, we can add any column with exactly n of the $2n$ bits set. This gives us a set of $2n$ by $2n + 1$ matrices that are row and column symmetries of each other. There are $(2n)!/(n!)^2$ bit vectors with exactly n of $2n$ bits set. Hence, we have $(2n)!/(n!)^2$ matrices which satisfy SNAKELEX that are in the same row and column symmetry class. Using Stirling's formula, this grows as $O(4^n/\sqrt{n})$. \square

8 Experimental Results

The proof of Theorem 1 gives a polynomial method to break all row and column symmetry. This allows us to compare symmetry breaking methods for matrix models like

DOUBLELEX and SNAKELEX, not only with respect to each other but for the first time in absolute terms. Our aim is to evaluate: first, whether the worst-case scenarios identified in theorems 2 and 8 are indicative of what can be expected in practice; second, how effective these methods are with respect to each other; third, in cases where they differ significantly, how much closer the best of them is to the optimal.

To answer these questions, we experimented with different symmetry breaking constraints: DOUBLELEX, the column-wise SNAKELEX (SNAKELEX_C) or the row-wise SNAKELEX (SNAKELEX_R) [16]. We use NOSB to denote no symmetry breaking constraints. For each problem instance we found the total number of solutions left by symmetry breaking constraints ($\#s$) and computed how many of them were symmetric based on the method outlined in the proof of Theorem 1. The number of *non symmetric* solutions is equal to the number of symmetry classes ($\#ns$) if the search space is exhausted. In all instances at least one model exhausted the search space to compute the of symmetry classes, shown in the column ROWWISELEX. We use ‘-’ to indicate that the search is not completed within the time limit. As the NOSB model typically could not exhaust the search space within the time limit, we use ‘>’ to indicate a lower bound on the number of solutions. Finally, we used a variable ordering heuristic that follows the corresponding lex-leader variable ordering in each set of symmetry breaking constraints (i.e. row-wise snake ordering with SNAKELEX_R). We ran experiments in Gecode 3.3.0 on an Intel XEON X5550, 2.66 GHz, 32 GB RAM with 18000 sec timeout.

Unconstrained problems. We first evaluated the effectiveness of symmetry breaking constraints in the absence of problem constraints. This gives the “pure” effect of these constraints at eliminating row and column symmetry. We considered a problem with a matrix $m_{r \times c}$, $r \leq c = [2, 6]$, $D(m_{r,c}) = [0, d - 1]$, $d = [2, \dots, 5]$ whose rows and columns are interchangeable. Table 1 summarizes the results. The first part presents typical results for 0/1 matrices whilst the second part presents results for larger domains. The results support the exponential worst case in Theorems 2 and 8 as the ratio of solutions found to symmetry classes increases from 1.25 (3,3,2) to over 6 (6,6,2), approximately doubling with each increase of the matrix size. As we increase the problem size, the number of symmetric solutions left by DOUBLELEX and SNAKELEX grows rapidly. Interestingly, SNAKELEX_C achieves better pruning on 0/1 matrices, while DOUBLELEX performs better with larger domains.

Constrained problems. Our second set of experiments was on three benchmark domains: Equidistant Frequency Permutation Array (EFPA), Balanced Incomplete Block Designs and Covering Array (CA) problems. We used the non-Boolean model of EFPA [4] (Table 2), the Boolean matrix model of BIBD [11] (Table 3) and a simple model of CA [17] (Table 4). We consider the satisfaction version of the CA problem with a given number of vectors b . In all problems instances the DOUBLELEX, SNAKELEX_R and SNAKELEX_C constraints show their effectiveness, leaving only a small fraction of symmetric solutions. Note that SNAKELEX_C often leaves fewer symmetric solutions. However, it is significantly slower compared to DOUBLELEX and SNAKELEX_R because it tends to prune later (thereby exploring larger search trees). For example, the number of failures for the (5, 3, 3, 4) EFPA problem is 21766, 14072 and 1129085 for DOUBLELEX, SNAKELEX_R and SNAKELEX_C respectively. On EFPA problems, SNAKELEX_R is about twice as fast as DOUBLELEX and leaves less solutions. On the

Table 1. Unconstrained problems. Number of solutions found by posting different sets of symmetry breaking constraints. r is the number of rows, c is the number of columns, d is the size of the domains.

(r, c, d)	RowWiseLEX	NOsB	DOUBLELEX	SNAKELEX _R	SNAKELEX _C
	#ns	#s	#s / time	#s / time	#s / time
(3, 3, 2)	36	512	45 / 0.00	44 / 0.00	44 / 0.00
(4, 4, 2)	317	65536	650 / 0.00	577 / 0.00	577 / 0.00
(5, 5, 2)	5624	3.36·10 ⁷	24520 / 0.05	18783 / 0.06	18783 / 0.06
(6, 6, 2)	251610	> 9.4·10 ⁹	2.62 · 10 ⁶ / 22.2	1.71 · 10⁶ / 22.2	1.71 · 10⁶ / 18.1
(3, 3, 3)	738	19683	1169 / 0.00	1232 / 0.00	1232 / 0.00
(3, 3, 4)	8240	2.62·10 ⁵	14178 / 0.03	15172 / 0.02	15172 / 0.05
(3, 3, 5)	57675	1.95·10 ⁶	1.02·10⁵ / 0.19	1.09·10 ⁵ / 0.15	1.09·10 ⁵ / 0.21
(3, 3, 6)	289716	1.01·10 ⁷	5.20·10⁵ / 2.32	5.54·10 ⁵ / 3.29	5.54·10 ⁵ / 2.83

Table 2. Equidistant Frequency Permutation Array problems. Number of solutions found by posting different sets of symmetry breaking constraints. v is the number code words, q is the number of different symbols, λ is the size of the domains.

(q, λ, d, v)	RowWiseLEX	NOsB	DOUBLELEX	SNAKELEX _R	SNAKELEX _C
	#ns	#s	#s / time	#s / time	#s / time
(3, 3, 2, 3)	6	1.81·10 ⁵	6 / 0.00	6 / 0.00	6 / 0.00
(4, 3, 3, 3)	8	> 3.88·10 ⁷	16 / 0.01	16 / 0.01	16 / 0.16
(4, 4, 2, 3)	12	> 5.87·10 ⁷	12 / 0.00	12 / 0.00	12 / 0.04
(3, 4, 6, 4)	1427	> 5.57·10 ⁷	11215 / 5.88	10760 / 5.36	8997 / 493.87
(4, 3, 5, 4)	8600	> 2.03·10 ⁷	61258 / 69.90	58575 / 51.62	54920 / 3474.09
(4, 4, 5, 4)	9696	> 5.45·10 ⁶	72251 / 173.72	66952 / 132.46	66168 / 14374.82
(5, 3, 3, 4)	5	> 4.72·10 ⁶	20 / 0.36	20 / 0.25	20 / 31.61
(3, 3, 4, 5)	18	> 2.47·10 ⁷	71 / 0.17	71 / 0.13	63 / 30.08
(3, 4, 6, 5)	4978	> 2.08·10 ⁷	77535 / 167.50	71186 / 137.88	—
(4, 3, 4, 5)	441	> 6.55·10 ⁶	2694 / 19.37	2688 / 12.80	2302 / 5960.43
(4, 4, 2, 5)	12	> 6.94·10 ⁶	12 / 0.02	12 / 0.01	12 / 1.60
(4, 4, 4, 5)	717	> 6.27·10 ⁶	4604 / 38.15	4397 / 24.58	—
(4, 6, 4, 5)	819	> 4.08·10 ⁶	5048 / 69.83	4736 / 44.83	—
(5, 3, 4, 5)	3067	> 2.39·10 ⁶	20831 / 403.97	20322 / 216.93	—
(6, 3, 4, 5)	15192	> 2.16·10 ⁶	1.11·10 ⁵ / 4924.41	1.06·10⁵ / 2006.19	—

Table 3. Balanced Incomplete Block Designs. Number of solutions found by posting different sets of symmetry breaking constraints. v is the number of objects, k is the objects in each block, every two distinct objects occur together in exactly λ blocks.

(v, k, λ)	RowWiseLEX	NOsB	DOUBLELEX	SNAKELEX _R	SNAKELEX _C
	#ns	#s	#s / time	#s / time	#s / time
(5, 2, 7)	1	> 0	1 / 0.01	1 / 0.02	1 / 73.26
(5, 3, 6)	1	> 1.51·10 ⁹	1 / 0.00	1 / 0.00	1 / 0.82
(6, 3, 4)	4	> 1.29·10 ⁹	21 / 0.01	25 / 0.00	21 / 12.62
(6, 3, 6)	6	> 1.21·10 ⁹	134 / 0.04	146 / 0.07	134 / 1685.58
(7, 3, 4)	35	> 1.18·10 ⁹	3209 / 0.33	9191 / 1.07	5270 / 7241.92
(7, 3, 5)	109	> 1.09·10 ⁹	33304 / 4.15	85242 / 11.90	—

Table 4. Covering Arrays. Number of solutions found by posting different sets of symmetry breaking constraints. b is the number of vectors, k is the length of a vector, g is the size of the domains, t is the covering strength.

(t, k, g, b)	RowWiseLEX	NOSB	DOUBLELEX	SNAKELEX _R	SNAKELEX _C
	#ns	#s	#s / time	#s / time	#s / time
(2, 3, 2, 4)	2	48	2 / 0.00	2 / 0.00	2 / 0.00
(2, 3, 2, 5)	8	1440	15 / 0.00	15 / 0.00	15 / 0.00
(2, 3, 3, 9)	6	$4.35 \cdot 10^6$	12 / 0.00	12 / 0.00	12 / 1.95
(2, 3, 3, 10)	104	$> 5.08 \cdot 10^8$	368 / 0.00	370 / 0.03	372 / 7.06
(2, 3, 3, 11)	1499	$> 5.56 \cdot 10^8$	6824 / 0.23	6905 / 0.24	6892 / 26.29
(2, 3, 4, 16)	150	> 0	576 / 0.72	576 / 0.70	—
(2, 3, 4, 17)	8236	> 0	43368 / 12.43	43512 / 12.82	—
(2, 3, 5, 25)	27280	> 0	$1.61 \cdot 10^5 / 1166.94$	$1.61 \cdot 10^5 / 1178.14$	—
(2, 4, 2, 5)	5	1920	10 / 0.00	10 / 0.00	10 / 0.00
(2, 4, 2, 7)	333	$1.60 \cdot 10^7$	2285 / 0.04	2224 / 0.07	1850 / 0.04
(2, 4, 3, 9)	5	$2.61 \cdot 10^7$	36 / 0.02	36 / 0.01	26 / 1102.30

CA problems DOUBLELEX and SNAKELEX_R show similar results, while DOUBLELEX performs better on BIBD problems in terms of the number of solution left.

Overall, our results show that DOUBLELEX and SNAKELEX prune most of the symmetric solutions. SNAKELEX_C slightly outperforms DOUBLELEX and SNAKELEX_R in terms of the number of solutions left, but it explores larger search trees and is about two orders of magnitude slower. However, there is little difference overall in the amount of symmetry eliminated by the three methods.

9 Other Related Work

Lubiw proved that any matrix has a row and column permutation in which rows and columns are lexicographically ordered and gave a nearly linear time algorithm to compute such a matrix [18]. Shlyakhter and Flener *et al.* independently proposed eliminating row and column symmetry using DOUBLELEX [10,11]. To break some of the remaining symmetry, Frisch, Jefferson and Miguel suggested ensuring that the first row is less than or equal to all permutations of all other rows [19]. As an alternative to ordering both rows and columns lexicographically, Frisch *et al.* proposed ordering the rows lexicographically but the columns with a multiset ordering [20]. More recently, Grayland *et al.* have proposed SNAKELEX, an alternative to DOUBLELEX based on linearizing the matrix in a snake-like way [16]. An alternative way to break the symmetry of interchangeable values is to convert it into a variable symmetry by channelling into a dual 0/1 viewpoint in which $Y_{ij} = 1$ iff $X_i = j$, and using lexicographical ordering constraints on the columns of the 0/1 matrix [1]. However, this hinders propagation [15]. Finally, dynamic methods like SBDS have been proposed to remove symmetry from the search tree [21]. Unfortunately, dynamic techniques tend not to work well with row and columns symmetries as the number of symmetries is usually too large.

10 Conclusions

We have provided a number of positive and negative results on dealing with row and column symmetry. To eliminate some (but not all) symmetry we can post static

constraints like DOUBLELEX and SNAKELEX. On the positive side, we proposed the first polynomial time method to eliminate *all* row and column symmetry when the number of rows (or columns) is bounded. On the negative side, we argued that DOUBLELEX and SNAKELEX can leave a large number of symmetric solutions. In addition, we proved that propagating DOUBLELEX completely is NP-hard. Finally, we showed that it is not always safe to combine Puget's value symmetry breaking constraints with row and column symmetry breaking constraints, correcting a claim made in the literature.

References

1. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetry in matrix models. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, p. 462. Springer, Heidelberg (2002)
2. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Matrix Modelling. Technical Report APES-36-2001, APES group, Presented at Formul 2001 (Workshop on Modelling and Problem Formulation), CP 2001 post-conference workshop (2001)
3. Crawford, J., Ginsberg, M., Luks, G., Roy, A.: Symmetry breaking predicates for search problems. In: Proceedings of 5th International Conference on Knowledge Representation and Reasoning (KR 1996), pp. 148–159 (1996)
4. Huczynska, S., McKay, P., Miguel, I., Nightingale, P.: Modelling equidistant frequency permutation arrays: An application of constraints to mathematics. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 50–64. Springer, Heidelberg (2009)
5. Bessiere, C., Hebrard, E., Hnich, B., Walsh, T.: The complexity of global constraints. In: Proceedings of the 19th National Conference on AI. AAAI, Menlo Park (2004)
6. Bessiere, C., Hebrard, E., Hnich, B., Walsh, T.: The complexity of global constraints. Constraints 12(2), 239–259 (2007)
7. Carlsson, M., Beldiceanu, N.: Arc-consistency for a chain of lexicographic ordering constraints. Technical report T2002-18, Swedish Institute of Computer Science (2002)
8. Katsirelos, G., Narodytska, N., Walsh, T.: Breaking Generator Symmetry. In: Proceedings of SymCon 2009 - 9th International Workshop on Symmetry and Constraint Satisfaction Problems, Colocated with CP 2009 (2009)
9. Puget, J.F.: Breaking symmetries in all different problems. In: Proceedings of 19th IJCAI, International Joint Conference on Artificial Intelligence, pp. 272–277 (2005)
10. Shlyakhter, I.: Generating effective symmetry-breaking predicates for search problems. Electronic Notes in Discrete Mathematics 9, 19–35 (2001)
11. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Symmetry in matrix models. Technical Report APES-30-2001, APES group. Presented at SymCon 2001 (Symmetry in Constraints), CP, post-conference workshop (2001)
12. Walsh, T.: Breaking Value Symmetry. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 880–887. Springer, Heidelberg (2007)
13. Puget, J.F.: Breaking all value symmetries in surjection problems. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 490–504. Springer, Heidelberg (2005)
14. Law, Y., Lee, J.: Global constraints for integer and set value precedence. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 362–376. Springer, Heidelberg (2004)
15. Walsh, T.: Symmetry breaking using value precedence. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) ECAI 2006, pp. 168–172. IOS Press, Amsterdam (2006)

16. Grayland, A., Miguel, I., Roney-Dougal, C.: Snake lex: An alternative to double lex. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 391–399. Springer, Heidelberg (2009)
17. Hnich, B., Prestwich, S., Selensky, E., Smith, B.: Constraint models for the covering test problem. *Constraints* 11, 199–219 (2006)
18. Lubiw, A.: Doubly lexical orderings of matrices. *SIAM J. on Computing* 16, 854–879 (1987)
19. Frisch, A., Jefferson, C., Miguel, I.: Constraints for breaking more row and column symmetries. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 318–332. Springer, Heidelberg (2003)
20. Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Multiset ordering constraints. In: Proceedings of 18th IJCAI, International Joint Conference on Artificial Intelligence (2003)
21. Gent, I., Smith, B.: Symmetry breaking in constraint programming. In: Horn, W. (ed.) Proceedings of ECAI-2000, pp. 599–603. IOS Press, Amsterdam (2000)

Ensemble Classification for Constraint Solver Configuration

Lars Kotthoff, Ian Miguel, and Peter Nightingale

University of St Andrews

{larsko,ianm,pn}@cs.st-andrews.ac.uk

Abstract. The automatic tuning of the parameters of algorithms and automatic selection of algorithms has received a lot of attention recently. One possible approach is the use of machine learning techniques to learn classifiers which, given the characteristics of a particular problem, make a decision as to which algorithm or what parameters to use. Little research has been done into which machine learning algorithms are suitable and the impact of picking the “right” over the “wrong” technique. This paper investigates the differences in performance of several techniques on different data sets. It furthermore provides evidence that by using a meta-technique which combines several machine learning algorithms, we can avoid the problem of having to pick the “best” one and still achieve good performance.

1 Introduction

The automatic selection of algorithms or parameters of algorithms is a problem that has been recognised for decades [19], but systematic investigation has only started relatively recently. Nowadays, systems that incorporate automatic selection of algorithms provide major performance improvements [17,22,12].

Most of these approaches use machine learning to uncover how the attributes of a particular problem affect the performance of a set of algorithms or a set of parameters for one algorithm. The designers of such systems face several difficult challenges. Which attributes are needed to capture the important effects on performance? Is the set of training instances representative and are the results likely to be applicable in general? Is the learned classifier overfitted?

The success or failure of such systems does not only depend on these factors, but also on the selection of an appropriate machine learning algorithm. This problem has received little attention so far – the choice is usually justified by the success of the system. But has the most appropriate technique been chosen and how likely is it that the chosen technique is also the best one in general?

We investigate this very question by applying many different machine learning techniques to two different algorithm selection problems. We propose an ensemble classification approach, which uses many different machine learning algorithms. We show that its performance is as good as and sometimes better than the performance of the best individual technique and at the same time more predictable and stable.

2 Background

The underlying problem is the algorithm selection problem, which was first described in [19]. Given a choice of algorithms and parameter settings, we want to choose the algorithm-parameter combination that delivers the best performance for a specific problem. Machine learning is an established approach for solving this problem, used for example in the PYTHIA [20] and MULTI-TAC [16] systems.

Two successful approaches in SAT are SATzilla [22] and SATenstein [12]. In CP, CP-Hydra uses a similar approach [17]. MULTI-TAC configured aspects of a solver based on instances. ACE [7] and Bain *et al* [2] learn search heuristics from instances. Genetic algorithms have been shown to be effective as well [1].

In machine learning, the combination of several classifiers is a well-established technique. In so-called ensemble learning [6], there are many different methods for creating different classifiers and combining their predictions, such as bootstrap aggregating, boosting and stacking [21].

The difference between different parameter settings for the same algorithms can have a more profound effect than choosing a different algorithm [14]. Selecting the most suitable machine learning algorithm and parameters for a set of instances is an area of active research in machine learning itself [4].

3 Algorithm Selection Data Sets

We investigate the performance of different machine learning algorithms on two algorithm selection problems. First, we decide whether to use g-nogood learning with lazy explanations [9] or not. Second, we consider the nine different versions of the alldifferent constraint detailed in [10]. These problems represent different important areas in constraint solver design. Lazy learning affects the search procedure, while the alldifferent constraint affects strength and efficiency of propagation. For lazy learning in particular, selecting different implementations instead of using a default one can provide a speedup of orders of magnitude.

We selected benchmark instances from Lecoutre’s XCSP repository [1] and from our own stock of problems, which includes many instances from previous CSP solver competitions. For lazy learning, we used 2028 problem instances from 46 different problem classes. Within a time limit of 5000 seconds, both the standard and the learning solvers were able to solve 1773 instances. For the alldifferent constraint, we used 1313 different instances from 16 different problem classes. We imposed a time limit of 3600 seconds; 1221 instances were solved by at least one of the candidate solvers within this limit. We took the median of three runs as the run time.

All experiments were run with binaries compiled with g++ version 4.4.3 and Boost version 1.40.0 on machines with 8 core Intel E5430 2.66GHz, 8GB RAM running CentOS with Linux kernel 2.6.18-164.6.1.el5 64Bit. Our reference solver is Minion [8] version 0.9. The binaries and instances required to reproduce the results are available from the authors on request.

¹ <http://tinyurl.com/y6hpphs>

4 Instance Attributes and Their Measurement

We measured 38 attributes of the problem instances. They describe a wide range of features such as constraint and variable statistics and a number of attributes based on the primal graph. The primal graph $g = \langle V, E \rangle$ has a vertex for every CSP variable, and two vertices are connected by an edge iff the two variables are in the scope of a constraint together.

Edge density. The number of edges in g divided by the number of pairs of distinct vertices.

Clustering coefficient. For a vertex v , the set of neighbours of v is $n(v)$. The edge density among the vertices $n(v)$ is calculated. The clustering coefficient is the mean average of this local edge density for all v .

Normalised degree. The normalised degree of a vertex is its degree divided by $|V|$. The minimum, maximum, mean and median normalised degree are used.

Normalised standard deviation of degree. The standard deviation of vertex degree is normalised by dividing by $|V|$.

Width of ordering. The width of a vertex v in an ordered graph, given by the variable ordering, is its number of *parents* (i.e. neighbours that precede v in the ordering). The width of the ordering is the maximum width over all vertices [5] and normalised by the number of vertices.

Width of graph. The width of a graph is the minimum width over all possible orderings, normalised by the number of vertices.

Variable domains. The quartiles and mean over the domains of all variables.

Constraint arity. The quartiles and the mean of the arity of all constraints (the number of variables constrained by it), normalised by the number of constraints.

Multiple shared variables. The proportion of pairs of constraints that share more than one variable.

Normalised mean constraints per variable. For each variable, we count the number of constraints on the variable. The mean average is taken, and this is normalised by dividing by the number of constraints.

Normalised SAC literals. The number of literals pruned by singleton consistency preprocessing, as a proportion of all literals.

Ratio of auxiliary variables to other variables. The ratio of auxiliary variables to other variables.

Tightness. The tightness of a constraint is the proportion of disallowed tuples. The tightness is estimated by sampling 1000 random tuples from the variable domains and testing if the tuple satisfies the constraint. The tightness quartiles and the mean over all constraints are used.

Proportion of symmetric variables. In many CSPs, the variables form equivalence classes where the number and type of constraints a variable is in are the same. The first stage of the algorithm used by Nauty [15] detects this property. Given a partition of n variables generated by this algorithm, we transform this into a number between 0 and 1 by taking the proportion of all pairs of variables which are in the same part of the partition.

Alldifferent statistics. The size of the union of all variables in an alldifferent constraint divided by the number of variables $|V|$. We used the quartiles and the mean over all alldifferent constraints.

We intended to cover a wide range of possible factors that affect the performance of the different algorithms with these attributes. We normalised attributes that would be specific to problem instances of a particular size. This is based on the intuition that similar instances of different sizes are likely to behave similarly with respect to the investigated algorithms. Computing the attributes took about 15 seconds per instance on average.

Not all attributes were applicable for both of the algorithm selection problems. For lazy learning, the alldifferent statistics did not apply. For alldifferent, we did not use the number of literals that SAC removes because it is different for different versions of the constraint.

5 Learning Classifiers

We annotated each benchmark instance with the algorithm variant that performed best on it based on the run times of the candidate algorithms for the specific algorithm selection problem. For the decision between the different implementations of the alldifferent constraint, we additionally considered the number of search nodes per second. If the problem instance was solved by no candidate within the timeout, we assigned the annotation “don’t know”. This data was given to the machine learning algorithms to learn a classifier that, given an instances, predicts which one of the algorithms will have the best performance on it.

We used the WEKA [11] machine learning software through the R interface to learn classifiers. We used almost all of the WEKA algorithms that were applicable to our problems – decision rules, decision trees, Bayesian classifiers, nearest neighbour and neural networks. Our selection is broad and includes almost all major machine learning methodologies. The specific classifiers we used are BayesNet, BFTree, ConjunctiveRule, DecisionTable, FT, HyperPipes, IBk, J48, J48graft, JRip, LADTree, MultilayerPerceptron, NBTree, OneR, PART, RandomForest, RandomTree and REPTree, all of which are described in [21].

We decided to measure the performance of the learned classifiers not in terms of the usual machine learning performance measures, but in terms of misclassification penalty [22]. The misclassification penalty is the additional CPU time we need to solve a problem instance if not choosing the optimal algorithm. This is based on the intuition that we do not particularly care about classifying as many instances correctly as possible; we rather care that the instances that are important to us are classified correctly. The higher the potential gain for an instance, the more important it is to us. If the selected algorithm was not able to solve the problem, we assumed the timeout value minus the CPU time the fastest algorithm took to be the misclassification penalty. This only gives a weak lower bound, but the correct value cannot be estimated easily.

We furthermore decided to assign the maximum misclassification penalty (or the maximum possible gain) as a cost to each instance to bias machine learning towards the instances we care about most. Each instance was attached a cost of $\max(1, 1 + \log_2(\text{penalty}))$. Note that we used the absolute and not the relative cost value – if the difference in absolute time is only 0.1 seconds, it does not matter if the relative difference is orders of magnitude.

To combine the different classifiers, we take the predictions of each classifier for an individual problem and choose the one that occurs most often; breaking ties by alphanumeric ordering. A thorough investigation in [3] showed that voting performs better in general than other techniques.

For each data set of the different algorithm selection problems, we generated partitions as follows. First, we removed the instances of a randomly selected problem class. Then we removed about 33% of the remaining instances at random. This data was used for training and the removed instances for testing. For both data sets, we generated 10 different partitions of approximately equal size this way.

The most important issue we are addressing is the generality of the learned classifier – given its performance on the data set we are using for testing, will it perform equally well on unknown data? There are two different cases. The unknown data could be new instances from a problem class which the classifier has seen before or the data could consist of unknown instances from unknown problem classes. We address both scenarios by removing individual problem classes and random instances from the original data set. Using this method, we test for overfitted classifiers at the same time.

We ran each machine learning algorithm on each training partition and evaluated its performance in terms of misclassification penalty through stratified 10-fold cross-validation [13]. The median of the 10 folds denotes our overall performance estimate. We then evaluated the performance of each of these classifiers on the respective test partition.

6 Results

Figure 1 shows the performance on the different partitions. It is obvious that the performance on one set of data, even when using cross-validation, is not a good predictor of the performance on another set of data, as shown by the length of the arrows. In only one of twenty cases, the classifier which performs best on the training partition is also the best one on the test partition. In two cases, the best classifier actually becomes the worst on new data. It also becomes clear that this effect is attenuated by using the ensemble classifier – in almost all cases, the performance differences on different sets of data are less pronounced, as denoted by the lengths of the arrows starting at the crosses (ensemble classifier) versus the ones starting at the circles (single classifier). The ensemble classifier is more robust in that its performance is predictable more reliably. The algorithm with the best average performance over all the data was **BFTree**; the difference to the ensemble classifier was about 1%.

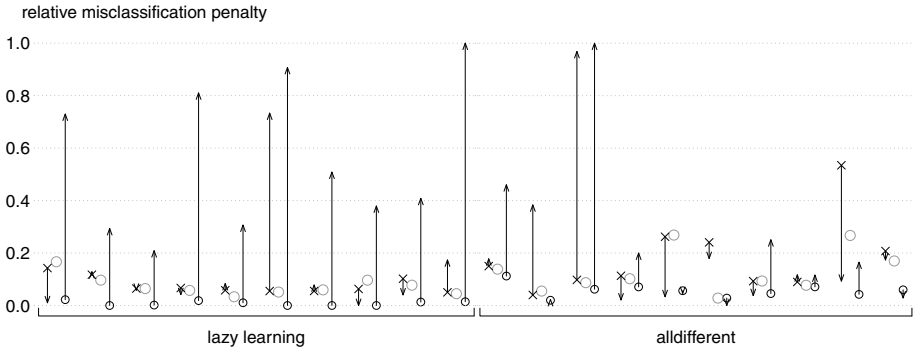


Fig. 1. Classifier performance on the different partitions. The misclassification penalties were normalised by the best classifier across all partitions (i.e. the misclassification penalty of the best classifier is always 1) and then scaled between best and worst classifier to make the different data sets comparable (i.e. the best classifier is now 0 and the worst 1). In absolute terms, the difference between best and worst is up to several orders of magnitude. The black circles show the performance of the classifier which performed best during cross-validation. The larger, grey circles show the performance of the classifier which was the best one on the test partition. The cross denotes the performance of the ensemble classifier during cross-validation. The end of the arrows denotes the performance on the test partition for best individual and ensemble classifiers. The length of the arrow denotes the uncertainty of the prediction of classifier performance from cross-validation.

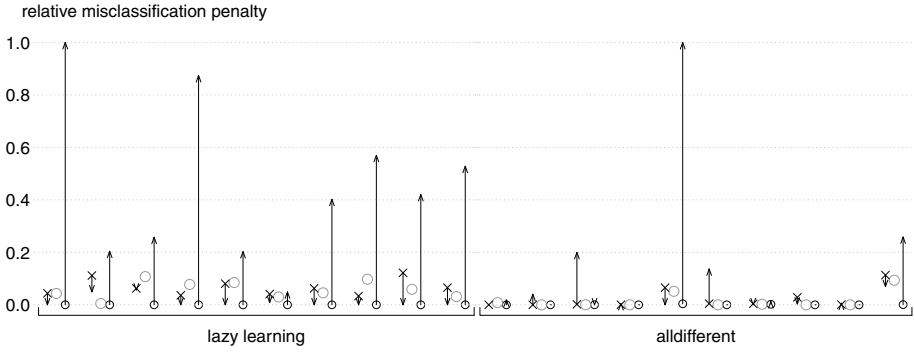


Fig. 2. Classifier performance for three different classifiers. Note that in five cases the performance of the ensemble classifier was better than that of an individual best one.

The figure furthermore shows that for several different partitions, our ensemble classifier performs better than the “best” individual algorithm on a single partition most of the time and is often close to or even better than the individual classifier that would be the best on unseen data. We achieve significant improvements without time-consuming inspection and evaluation of individual machine learning algorithms to select the most suitable one and tune its parameters.

Our results do not depend on a large number of machine learning algorithms. Figure 2 shows the results with just three algorithms from different machine learning methodologies – `BayesNet`, `MultilayerPerceptron` (a neural network algorithm) and `J48` (an implementation of the well-known C4.5 algorithm [18]). The improvements over using a single classifier are comparable to the ones shown in Figure 1. In some cases we even achieve an improvement over the best individual classifier through the combination of the predictions of several classifiers. Note that these three individual algorithms were not selected because of particularly good overall performance – none of them was the best on average and they often performed worse than the ensemble classifier.

In terms of solve time, the ensemble classification approach improves over always making a default decision. The improvement is substantial for lazy learning and marginal for all different.

7 Conclusions

We have presented a thorough and in-depth investigation into the variability of the performance of different machine learning algorithms and techniques on two real-world algorithm selection problems. We based our investigation on experimental results for a large number of diverse problems and a large number of different machine learning techniques. Although much research has been done in the field of algorithm selection and algorithm tuning, no similar evaluation of the methodology has been undertaken, to the best of our knowledge.

Our results conclusively show that the performance of a machine learning algorithm is so variable that predictions as to its generality and performance on new data cannot be made without investing significant effort into substantiating these claims. Furthermore, an algorithm which may have a low performance and therefore appear unsuitable on test data has the potential for performing much better on unknown data.

The technique we are proposing for the configuration of constraint solvers, ensemble classification by combining the predictions of several classifiers by majority vote, improves on this. Our experiments provide strong evidence that its performance on several data sets will in general be better than the performance of an individual best classifier on one data set. Indeed it will be close to the performance achieved by the classifier in the ensemble which is the best for a given data set. We furthermore observed cases in which the ensemble classifier was better than a single classifier even on a single data set.

While combining several classifiers adds significant overhead in the offline phase when more classifiers need to be learned, the overhead in the online phase when new instances are classified was negligible in our experiments. The time required to compute the instance attributes was much higher than the time for running additional classifiers and combining their predictions in all cases. In particular, running an individual classifier on a single problem instance only takes a few milliseconds on average.

The main advantage of ensemble classification is that individual machine learning algorithms can be combined without intrinsic knowledge about each one of them. The level of machine learning expertise required is reduced significantly without affecting the results significantly. Note that this does not mitigate the need for domain knowledge to select relevant features for example. Ensemble classification enables practitioners without a lot of machine learning knowledge to apply machine learning to their problems.

Acknowledgements

We thank Chris Jefferson for the description of one of the problem attributes used in the analysis, Jesse Hoey for useful discussions about machine learning, and anonymous reviewers for their feedback. Lars Kotthoff is supported by a SICSA studentship. This work was supported by EPSRC grant EP/H004092/1.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009)
2. Bain, S., Thornton, J., Sattar, A.: Evolving Variable-Ordering heuristics for constrained optimisation. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 732–736. Springer, Heidelberg (2005)
3. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Mach. Learn.* 36(1-2), 105–139 (1999)
4. Chen, F., Jin, R.: Active algorithm selection. In: AAI, pp. 534–539 (2007)
5. Dechter, R.: *Constraint Processing*. Elsevier Science, Amsterdam (2003)
6. Dietterich, T.G.: Ensemble methods in machine learning. In: *First International Workshop on Multiple Classifier Systems*, pp. 1–15 (2000)
7. Epstein, S., Freuder, E., Wallace, R., Morozov, A., Samuels, B.: The adaptive constraint engine. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 525–542. Springer, Heidelberg (2002)
8. Gent, I., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: ECAI, pp. 98–102 (2006)
9. Gent, I.P., Miguel, I., Moore, N.C.A.: Lazy explanations for constraint propagators. In: Carro, M., Peña, R. (eds.) PADL 2010. LNCS, vol. 5937, pp. 217–233. Springer, Heidelberg (2010)
10. Gent, I., Miguel, I., Nightingale, P.: Generalised arc consistency for the alldifferent constraint: An empirical survey. *Artif. Intell.* 172(18), 1973–2000 (2008)
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: An update. *SIGKDD Explorations* 11(1) (2009)
12. KhudaBukhsh, A., Xu, L., Hoos, H., Leyton-Brown, K.: SATenstein: Automatically building local search SAT solvers from components. In: IJCAI, pp. 517–524 (2009)
13. Kohavi, R.: A study of Cross-Validation and bootstrap for accuracy estimation and model selection. In: IJCAI, p. 1137–1143 (1995)
14. Lavesson, N., Davidsson, P.: Quantifying the impact of learning algorithm parameter tuning. In: AAI, pp. 395–400 (2006)

15. McKay, B.: Practical graph isomorphism. In: Numerical Mathematics and Computing, pp. 45–87 (1981)
16. Minton, S.: Automatically configuring constraint satisfaction programs: A case study. *Constraints* 1(1/2), 7–43 (1996)
17. O’Mahony, E., Hebrard, E., Holland, A., Nugent, C., O’Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: Irish Conf. on AI (2008)
18. Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco (1993)
19. Rice, J.: The algorithm selection problem. *Adv. Computers* 15, 65–118 (1976)
20. Weerawarana, S., Houstis, E.N., Rice, J.R., Joshi, A., Houstis, C.E.: PYTHIA: a knowledge-based system to select scientific algorithms. *ACM Trans. Math. Softw.* 22(4), 447–468 (1996)
21. Witten, I., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco (2005)
22. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. *JAIR* 32, 565–606 (2008)

On Testing Constraint Programs

Nadjib Lazaar¹, Arnaud Gotlieb¹, and Yahia Lebbah²

¹ INRIA Rennes Bretagne Atlantique, Campus Beaulieu, 35042 Rennes, France
{nadjib.lazaar, arnaud.gotlieb}@irisa.fr

² Université d'Oran Es-Senia, Lab. LITIO, B.P. 1524 EL-M'Naouar,
31000 Oran, Algeria
Université de Nice-Sophia Antipolis, I3S-CNRS, France
ylebbah@gmail.com

Abstract. The success of several constraint-based modeling languages such as OPL, ZINC, or COMET, appeals for better software engineering practices, particularly in the testing phase. This paper introduces a testing framework enabling automated test case generation for constraint programming. We propose a general framework of constraint program development which supposes that a first declarative and simple constraint model is available from the problem specifications analysis. Then, this model is refined using classical techniques such as constraint reformulation, surrogate and global constraint addition, or symmetry-breaking to form an improved constraint model that must be thoroughly tested before being used to address real-sized problems. We think that most of the faults are introduced in this refinement step and propose a process which takes the first declarative model as an oracle for detecting non-conformities. We derive practical test purposes from this process to generate automatically test data that exhibit non-conformities. We implemented this approach in a new tool called CPTEST that was used to automatically detect non-conformities on two classical benchmark programs, namely the Golomb rulers and the car-sequencing problem.

1 Introduction

Constraint programs such as those written in modern Constraint Programming languages and platforms (e.g. OPL¹, COMET², ZINC³, CHOCO⁴, GECODE⁵, ...), aim at solving industrial combinatorial problems that arise in optimization, planning, or scheduling. Recently, a new trend has emerged that propose also to use CP programs to address critical applications in e-Commerce⁵, air-traffic control and management^{3,6}, and critical software development¹⁴. While constraint program debugging drew the attention of some researchers, few supports in terms of software engineering and testing have been proposed to

¹ www.ilog.com/products/oplstudio/

² www.dynadec.com/support/downloads/

³ www.g12.cs.mu.oz.au/

⁴ choco.sourceforge.net

⁵ www.gecode.org

help verify critical constraint programs. Automatic debugging of constraints programs has been an important topic of the OADymPPaC⁶ project, that resulted in the definition of generic trace models [2,7], the development of post-mortem trace analyzers, such as Codeine for Prolog, Morphine [7] for Mercury, ILOG Gentra4CP, or JPalm/JChoco. These models and tools help understand constraint programs and contribute to their optimization and correction, but they are not dedicated to systematic fault detection. Indeed, functional fault detection requires the definition of a reference (called an oracle in software testing) in order to check the conformity between an implementation and its reference [11]. Automatic fault detection also requires the definition of test purpose to decide when to stop testing [12]. Whereas conventional software development benefits from research advances in software verification (including static analysis, model checking or automated test data generation), developers of constraint programs are still confined to perform systematic verification by hand.

Automatic constraint program testing cannot be easily handled by existing testing approaches because of the two following reasons: firstly, constraint programs are intrinsically non-deterministic as they represent sets of solutions and conventional definitions of conformity do not apply ; secondly, the refinement process of constraint programs is specific to CP. Indeed, developers usually start with an initial declarative constraint model of the problem, which faithfully translates the problem specifications, without granting interest to its performances. As this model cannot handle large-sized instances of the problem, they exploit several refinement techniques to build an improved model. For example, usual refinement techniques include the use of dedicated data structures, constraint reformulation, global constraints addition, redundant and surrogate constraint addition, as well as constraints which break symmetries (these constraints usually improve considerably the effectiveness of the solving process). The refinement process, carried out by the developer, is an error-prone process and we believe that most of the faults are introduced during this step.

In this article, we propose a testing framework for checking the correctness of a constraint program implementation. The oracle for the constraint program under test is an initial declarative model considered to be valid w.r.t. the user requirements. Our framework is based on the definition of four distinct conformity relations to handle constraint satisfaction problems as well as optimization problems. A practical consequence of these definitions is the proposal of test purposes for evaluating the conformance of constraint programs. Note that this paper does not address another essential topic of CP verification which is the correction of solvers or optimizers. We propose an algorithm for checking the correction of the CP program under test that solves a set of derived constraint problems able to exhibit non-conformities. We implemented our approach in a tool called CPTEST that seeks non-conformities in OPL programs. For evaluating the proposed testing process, CPTEST was used to find non-conformities in various faulty OPL constraint programs of the Golomb rulers and

⁶ Contraintes/OADymPPaC/

the car-sequencing problem. It was also used to assess the conformity for small instances of the problem.

The rest of the paper is organized as follows: Sec. 2 illustrates our testing framework on a simple case in order to show a typical non-conformity case. Sec. 3 gives the definition of conformity relations required in the framework. In Sec. 4, the testing process we derive from these definitions is introduced and illustrated on a simple example. Sec. 5 presents the CPTEST tool and details our experimental evaluation. Finally, Sec. 6 concludes the paper and draws some perspectives to this work.

2 An Illustrative Example

Let us illustrate some of the refinement techniques on the classical problem of the Golomb rulers, which has various applications in fields such as Radio communications or X-Ray crystallography.

A Golomb ruler [8] is a set of m marks $0 = x_1 < x_2 < \dots < x_m$ such as $m(m-1)/2$ distances $\{x_j - x_i \mid 1 \leq i < j \leq m\}$ are distinct. A ruler is of order m if it contains m marks, and it is of length x_m . The goal is to find a ruler of order m with minimal length (*minimize* x_m). A declarative model of this problem is given in part A of Fig. 1 while part B presents a refined and improved model. It is easy to convince a human that model A actually solves the Golomb rulers problem, but this is much more difficult for model B. Indeed, model

<pre> int m=...; dvar int+ x[1..m]; minimize x[m]; subject to { c1: forall (i in 1..m-1) x[i] < x[i+1]; c2: forall (i,j,k,l in 1..m : (i < j && k < l && (i != k j != l))) x[j] - x[i] != x[l] - x[k]; } </pre> <p style="text-align: center;">- A -</p>	<pre> int m=...; dvar int x[1..m] in 0..m*m; tuple indexerTuple { int i; int j;} {indexerTuple} indexes={<i,j> i,j in 1..m: i < j}; dvar int d[indexes]; minimize x[m]; subject to { cc1: forall (i in 1..m-1) x[i] < x[i+1]; cc2: forall(ind in indexes) d[ind] == x[ind.i]-x[ind.j]; cc3: x[1]=0; cc4: x[m] >= (m * (m - 1)) / 2; // cc5: allDifferent(all(ind in indexes) d[ind]); cc6: x[2] <= x[m]-x[m-1]; cc7: forall(ind1 in indexes, ind2 in indexes, ind3 in indexes: (ind1.i==ind2.i)&& (ind2.j==ind3.j) &&(ind1.j==ind3.j)&& (ind1.i<ind2.j < ind1.j)) d[ind1]==d[ind2]+d[ind3]; cc8: forall(ind1,ind2,ind3,ind4 in indexes: (ind1.i==ind2.i)&&(ind1.j==ind3.j)&& (ind2.j==ind4.j)&&(ind3.i==ind4.i)&&(ind1.i<m-1) &&(3<ind1.j<m+1)&&(2<ind2.j<m)&&(1<ind3.i<m-1)&& (ind1.i < ind3.i < ind2.j < ind1.j)) d[ind1]==d[ind2]+d[ind3]-d[ind4]; cc9: forall(i in 2..m, j in 2..m, k in 1..m : i < j) x[i]=x[i-1]+k => x[j] != x[j-1]+k; } </pre> <p style="text-align: center;">- B -</p>
--	--

Fig. 1. $M_x(k)$ and $P_x(k)$ of Golomb rulers problem in OPL

B uses a matrix as data structure (`d[indexes]`), statically breaks symmetries (`cc6`), it contains redundant and surrogate constraints (`cc7`, `cc8`, `cc9`) and global constraints (`allDifferent`). In this paper, we address the fundamental question of revealing non-conformities in between the constraint program under test B and the model-oracle A. Testing B before using it on large instances of the problem (when $m > 15$) is highly desirable as computing the global minimum of the problem for these instances may require computation time greater than a week. Note that B is syntactically correct and provides correct Golomb rulers for small values of m . Our testing framework tries to find an instantiation of the variables that satisfies the constraints of B and violates at least one constraint of A. This testing process is detailed in section 4. With $m = 8$, our CPTEST framework computes $x = [0\ 1\ 3\ 6\ 10\ 26\ 27\ 28]$ in less than 6sec on a standard machine, indicating that B does not conform A and then contains a fault. Indeed, x is not a Golomb ruler as $27 - 26 = 1 - 0 = 1$. In fact, this non-conformity can easily be tackled by removing the comment on constraint `cc5` in part B. Doing so; CPTEST provides a conformity certificate saying that the CP program actually computes the global minimum in 10034.69sec (about 3hours). However, note that this certificate is only valid for $m = 8$. Note also that our framework can handle non-conformities of the Golomb rulers where the global minimum requirement is relaxed in order to deal with larger instances (when $m > 30$).

3 Testing Constraint Programs

3.1 Notations

In the rest of the paper, x denotes a vector of variables and $(x \setminus x_i)$ stands for substituting x by the valuation x_i .

A constraint program includes a constraint model $M_x(k)$, which is a conjunction of constraints $C_i(x)$ over variables x parameterized by k , the parameters vector of the model. Note that x may depend on k . For the Golomb rulers, k is the order of the ruler while x represents the vector of marks. If $k = 3$ then one seeks for a ruler with 3 marks (e.g., $\mathbf{x}=[0\ 1\ 3]$) while if $k = 4$ one seeks for a ruler with 4 marks (e.g., $\mathbf{x}=[0\ 1\ 4\ 6]$). *Solve()* is a generic procedure representing either the call to a constraint solver in the case of constraint satisfaction problem or the call to an optimization procedure. In this latter case, we note f the cost function (for the sake of clarity, f will be a minimization function but maximization problems can be tackled as well). We consider that k belongs to \mathcal{K} the set of possible values of the parameters for which $M_x(k)$ has at least one solution. $sol(M_x(k))$ denotes the set of solutions of $M_x(k)$ and while $Proj_y(sol(M_x(k)))$ expresses the projection of $sol(M_x(k))$ on the set y when $y \subseteq x$. In optimization problems, one usually starts with feasible solutions ranging in a cost interval $[l, u]$. Therefore, we introduce the set

$$Bounds_{f,l,u}(M_x(k)) = \{x | x \in sol(M_x(k)), f(x) \in [l, u]\}$$

$$\begin{array}{l} \text{Model } M_x(k) \\ \left\{ \begin{array}{l} C_1(x) \\ \dots \\ C_n(x) \\ \text{Solve}() \end{array} \right. \end{array}$$

To clarify these notations, Fig. 2 shows an example of a real objective function where point x_1 is a global minimum with a cost $f(x_1) = b$ and points x_0, x_3 belongs to $Bounds_{f,l,u}(M_x(k))$. Note that x_1 as well as x_2 do not necessarily belong to $Bounds_{f,l,u}(M_x(k))$.

3.2 Constraint Models and Programs

In our framework, we consider the initial declarative constraint model to be a testing oracle, called the *Model – Oracle*, and noted $M_x(k)$. $M_x(k)$ represents all the solutions of the problem and strictly conforms the problem specifications. We suppose that, for any parameter instantiation, $M_x(k)$ possesses at least one solution. Considering unsatisfiable Model–Oracles could be interesting for some applications (such as software verification [4]) but we excluded these cases in order to avoid considering equivalence of unsatisfiable models. The *Constraint Program Under Test (CPUT)* is a constraint model $P_z(k)$ (possibly unsatisfiable) which has to be tested for correction against the Model–Oracle. $P_z(k)$ is intended to solve difficult instances of the problem. We built our framework on the hypothesis that checking whether $M_{(x \setminus x_0)}(k_0)$ is true where x_0 is a point of the search space is not hard, while finding such an x_0 satisfying the constraints may be hard. Given a CPUT $P_z(k)$ and its Model–Oracle $M_x(k)$, we suppose that $x \subseteq z$ as $P_z(k)$ was obtained by refining $M_x(k)$. Hence, the set of variables in z distinct of x are dependant variables that are automatically instantiated when x is instantiated.

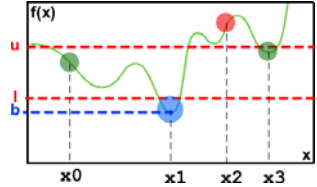


Fig. 2. Objective solutions

3.3 Conformity Relations

The correction of a CPUT w.r.t. a Model–Oracle can be approached through the usage of conformity relations. These relations aim at assessing the correction of the CPUT, a notion that can be expressed with various levels of depth. We propose four set-based definition of conformity divided on two groups: conformity relations adapted to constraint satisfaction problems and conformity relations for optimization problems.

Conformity relations for constraint satisfaction problems. The simplest definition of correction, well-adapted for problems where a single solution is sought, is given by the following conformity relation:

Definition 1 (*conf_{one}*)

$$\begin{aligned}
 P \text{ conf}_{one}^k M &\Leftrightarrow Proj_x(sol(P_z(k))) \neq \emptyset \wedge Proj_x(sol(P_z(k))) \subseteq sol(M_x(k)) \\
 P \text{ conf}_{one} M &\Leftrightarrow (\forall k \in \mathcal{K}, P \text{ conf}_{one}^k M)
 \end{aligned}$$

Roughly speaking, for a given instance k , $conf_{one}^k$ asks the solutions of the CPUT to be included in the solutions of the Model–Oracle. As an example,

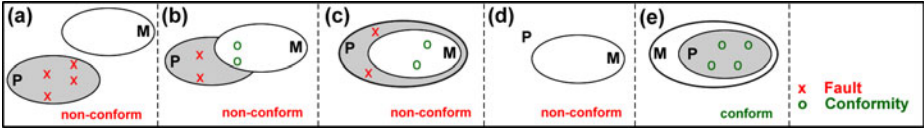


Fig. 3. $conf_{one}$ on $P_z(k)$ and $M_x(k)$

Fig. 3 presents both the sets $sol(M_x(k))$ noted M and $sol_x(P_z(k))$ noted P, where points in red **x** raise non-conformities (i.e., faults in the CPUT) while points in green **o** are conform w.r.t. the Model–Oracle. Parts (a)(b)(c) of Fig. 3 exhibit non-conformities as solving $P_z(k)$ can lead to solutions which do not satisfy $M_x(k)$. Part (d) does not exhibit any non-conformity but, as P does not contain any solution, it does not conform the Model–Oracle for $conf_{one}$. This example also shows that unsatisfiable models must be considered as non-conform w.r.t. Model–Oracles, in order to tackle faulty unsatisfiable CPUTs. On the contrary, part (e) of Fig. 3 shows that $P_z(k)$ conforms $M_x(k)$ for $conf_{one}$, as P cannot contain any non-conformity points.

Whenever all the solutions are sought, another definition of conformity is useful:

Definition 2 ($conf_{all}$)

$$\begin{aligned}
 P \text{ conf}_{all}^k M &\Leftrightarrow Proj_x(sol(P_z(k))) = sol(M_x(k)) (\neq \emptyset) \\
 P \text{ conf}_{all} M &\Leftrightarrow (\forall k \in \mathcal{K}, P \text{ conf}_{all}^k M)
 \end{aligned}$$

Roughly speaking, $conf_{all}$ asks for both set of solutions to be the same. Satisfying this conformity relation is very demanding and not always pertinent. For instance, the CPUT in part B of Fig. 1 includes constraints that break symmetries of the problem (e.g., cc6), which yields to lose solutions from the Model-Oracle. As a result, those two models cannot be conform w.r.t. $conf_{all}$.

In Fig. 4, parts (a)(b)(c) and (d) exhibit non-conformities. Part (d) shows a solution of the Model–Oracle which is not solution of the CPUT ; therefore, the CPUT is a faulty over-constrained model. Part (c) exhibits the opposite case where the CPUT is a faulty under-constrained model. Proving that $P_z(k)$ conforms $M_x(k)$ for one of these two conformity relations is highly desirable. Unfortunately, such a proof would require not only to find all the solutions of the CPUT which is an NP-hard problem for some constraint languages (e.g., the

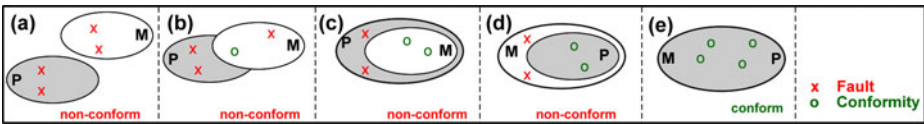


Fig. 4. $conf_{all}$ on $P_z(k)$ and $M_x(k)$

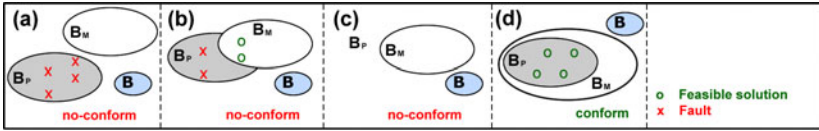


Fig. 5. $conf_{bounds}$ on $P_x(k)$ and $M_x(k)$

finite domains constraint language), but also to perform this for any value of k . This seems to be intractable in general (probably undecidable) and then we will confine ourselves to the search of non-conformities within finite resources.

Conformity relations for optimization problems. Conformity relations for optimization problems is harder to define, as practitioners usually start their refinement process by the definition of bounds for the optimal case [9]. Note also that non-conformities may arise in the cost function itself and we wanted our conformity relations to be able to tackle those cases.

Fig 5 presents the conformity relation where feasible solutions of the CPUT are sought in $[l', u']$. B_P denotes the set $Bounds_{f', l', u'}(P_x(k))$, B_M denotes the set $Bounds_{f, l, u}(M_x(k))$ while B is the set of global minima of $M_x(k)$. Part (a) exhibits four non-conformities as these points are not feasible solutions of the Model-Oracle $M_x(k)$ in $[l, u]$. For the same reason, Part (b) exhibits two non-conformities as two feasible solutions of B_P with cost in $[l', u']$ do not belong to B_M . Part (c) presents also a non-conformity as B_P does not contain any feasible point meaning that the minimization problem cannot find a feasible solution with cost in $[l', u']$. On the contrary, part (d) shows conformity because solutions of B_P belong to B_M . Formaly speaking,

Definition 3 ($conf_{bounds}$)

$$P \text{ } conf_{bounds}^k \text{ } M \Leftrightarrow Proj_x(bounds_{f', l', u'}(P_z(k))) \neq \emptyset \wedge Proj_x(bounds_{f', l', u'}(P_z(k))) \subseteq bounds_{f, l, u}(M_x(k))$$

Note that the definition of $conf_{bounds}$ does not require that $f = f'$ and then cases where the cost function has been refined can also be handled. This conformity relation is useful for addressing hard optimization problems as it does not require the computation of global minima. As a result, it can be used to assess the correction of models on relaxed instances of the global optimization problems. We will come back on this advantage in the experimental validation section. However, for some problems, it may be useful to assess not only the correction but also the fact that the CPUT actually computes optimal solutions. This can be performed by using the following definition which ensures that the global optimum belongs to $[l', u']$.

Definition 4 ($conf_{best}$)

$$P \text{ } conf_{best}^k \text{ } M \Leftrightarrow \begin{cases} P \text{ } conf_{bounds}^k \text{ } M, \\ bounds_{f, -\infty, l}(M_x(k)) = \emptyset, \\ bounds_{f', -\infty, l'}(P_z(k)) = \emptyset \end{cases}$$

4 A CP Testing Framework

Testing a CPUT w.r.t. an model-oracle requires to select test data. In this context, a test datum defines an instance of the CPUT and a point of the search space.

Definition 5 (Test datum). *Given a CPUT $P_z(k)$ and a Model-Oracle $M_x(k)$, a test datum is an instantiated pair (k_0, x_0) of parameters and variables.*

Note that evaluating $M_k(x)$ on the test datum (k_0, x_0) results true when x_0 is a solution of the model and false otherwise. Test execution is realized by evaluating both $P_{z \setminus z_0}(k_0)$ and $M_{x \setminus x_0}(k_0)$ and checks whether the results (either true or false) are the same. Depending on the selected conformity relation, a test verdict can be issued. This elementary process can be repeated as long as one wishes, but it is more interesting to guide the test data generation process by the use of *test purposes*. Seeking non-conformities implies finding test data such as the CPUT is satisfied and the Model-Oracle is violated. This enables to detect faults in CPUT, and helps the constraint programmer to revisit its refinements. Based on the selection of a conformity relation, non-conformities can be sought with the following test purposes:

conf_{one}. Given k , find a solution to $P_z(k) \wedge \neg C_i$ where C_i is a constraint of the Model-Oracle $M_x(k)$. The idea here is to isolate a non-conformity by looking independently at each constraint of the model-oracle. Considering all the constraints of the model-oracle would also be possible but less efficient to detect non-conformities as more constraints would be involved. Note that heuristics can be defined on the order of constraints to consider first. Note also that proving the unsatisfiability of $P_z(k) \wedge \neg C_i$ for all $C_i \in M_x(k)$ permits to issue a *conformity certificate* saying that $P \text{ conf}_{one}^k M$.

conf_{all}. Given k , find a solution to $(M_x(k) \wedge \neg C'_i) \vee (P_z(k) \wedge \neg C_i)$ where C_i (resp. C'_i) is a constraint of the Model-Oracle $M_x(k)$ (resp. $P_z(k)$). In this case, proving the unsatisfiability of these constraints permits to issue the conformity certificate $P \text{ conf}_{all}^k M$, but this is not often desirable as constraint solving usually requires to issue a single solution instead of all solutions.

conf_{bounds}. Given k and $[l', u']$, find a solution to $P_z(k) \wedge \neg C_i \wedge f'(z) \in [l', u'] \wedge f(x) \in [l, u]$ where f, f' are the cost functions of the Model-Oracle $M_x(k)$ and the CPUT $P_z(k)$. Proving that these constraints are unsatisfiable permits to issue a certificate $P \text{ conf}_{bounds}^k M$.

conf_{best}. Given k , find a solution to $(P \neg \text{conf}_{bounds}^k M) \vee \text{bounds}_{f, -\infty, l}(M_x(k)) \neq \emptyset \vee \text{bounds}_{f', -\infty, l'}(P_z(k)) \neq \emptyset$. Proving that these constraints are unsatisfiable permits to issue a conformity certificate $P \text{ conf}_{best}^k M$.

Interestingly, any solution found by the guidance of one of these test purposes can be stored for further investigations. Indeed, it can be used to debug the CPUT by looking at the violated constraint and it can also enrich a test set that will serve to assess the correction of future versions of the CPUT. In addition,

⁷ z_0 is obtained by extending x_0 with values depending on x_0 .

conformity certificates are essential for those who want to convince third-party certification authorities that their CP programs can be used in critical systems [54]. So, the proposed testing framework has a role to play in various phases of the constraint program development.

We now propose a simple but generic algorithm for searching non-conformities:

Algorithm 1. `one_negated($B, \{C_1, \dots, C_n\}$)`

Input : $B, \{C_1, \dots, C_n\}$ sets of constraints.

Output: *conf* when $\{C_1, \dots, C_n\}$ conform B , $\neg\text{conf}$ (+ non-conformity point) otherwise

$nc \leftarrow \emptyset$

$X \leftarrow \text{vars}(B)$

foreach $C_i \in \{C_1, \dots, C_n\}$ **do**

$V \leftarrow \text{vars}(C_i)/X$

if $V = \emptyset$ **then** $nc \leftarrow \text{Solve}(B \wedge \neg C_i)$

else $nc \leftarrow \text{Solve}(B \wedge \neg \text{Proj}_X(C_i))$

if nc **then return** $\neg\text{conf}(nc)$

end

return *conf*

where $\text{Solve}(B)$ denotes the algorithm to find the first solution of the constraints B , $\text{vars}(B)$ denotes the set of variables in B and $\text{Proj}_X(C)$ denotes the constraint projection on variables X .

Algorithm 1 takes two constraint sets as input and returns either *conf* when both sets conform with relation *conf_{one}* or $\neg\text{conf}$ (non-conformity point) where a non-conformity point has been found. Note that the other conformity relations can easily be implemented using this algorithm just by adjusting the call parameters. Special care has to be taken when building the negation of a model. For example, consider a Model-Oracle M with $x-y \neq x-z$; $x-y = y-z$; $x-z = y-z$; and a CPUT P with $c1: x-y=d1$; $c2: x-z=d2$; $c3: y-z=d3$; $c4: \text{allDiff}(d1, d2, d3)$; . Here, it is trivial to see that $P \text{ conf}_{all} M$ but if $c1$ is selected for negation, $M \wedge \neg c1$ has solutions as $d1$ is out of the scope of M . In the definitions of the conformity relations, these cases were discarded by the use of projections on the variables of the model-oracle. As computing general projections are expensive, improvements and pragmatic solutions are available in our implementation (see Sec. 5).

Providing that the underlying constraint solver is sound and complete, this algorithm is sound as it cannot report *conf* if there exists a non-conformity point. Indeed, given k , upon completion of the algorithm the unsatisfiability of $P_z(k) \wedge \neg M_x(k)$ is demonstrated showing that both models conform with the selection conformity relation. It is also complete as it cannot report false non-conformities.

A keypoint of our approach is that test data can be automatically generated using the same constraint solver as the one used for solving the CPUT. Recall that we rely on the solver and we are only interested in detecting non-conformities in models.

5 Experimental Validation

5.1 Implementation

We implemented the testing framework shown above in a tool called CPTEST for OPL (Optimization Programming Language [10]). We chose OPL because it is one of the main programming environments for developing constraint programs and also critical constraint programs [3]. CPTEST is based on ILOG CP Optimizer 2.1 from ILOG OPL 6.1.1 Development Studio. All our experiments were performed on Quadcore IntelXeon 3.16Ghz machine with 16GB of RAM and all the models we used to perform these experiments are available online at www.irisa.fr/celtique/lazaar/CPTEST.

CPTEST includes a complete OPL parser and a backend process that produces dedicated OPL programs as output. These OPL programs must be solved in order to find non-conformities. If a solution is found, then CPTEST stops and reports the non-conformity to the user. Whenever all these OPL programs are shown to be inconsistent, then a conformity certificate is issued. The tool is parameterized by several options, including the chosen conformity relation, the instance of the problem, etc. CPTEST handles the overall OPL language and can negate most of the constraints that can be expressed in OPL. However, it cannot negate all the global constraints available, such as the `cumulative` or `circuit` global constraint. Tab. 1 summarizes the syntax of OPL constraints handled by CPTEST. OPL includes two aggregators, namely `forall` and `or`. The universal qualifier `forall` is used to declare a collection of closely related constraints and to build global constraints. Interestingly, the `or` aggregator can be used to negate `forall`, as `or` implements existential quantification. The OPL `If-then-else` statement is less general than it may appear as its condition cannot contain decision variables. Its negation can be computed by negating the `Then`-part and `Else`-part without any loss of generality, as our goal is only to find non-conformities instead of computing the negation of a general model. Our CPTEST tool handles several global constraints over discrete values, namely `allDifferent`, `allMinDistance`, `inverse`, `forbiddenAssignments`, `allowedAssignments` and `pack`. These constraints can be represented as an aggregation of constraints and then computing their negation becomes trivial with the rules presented above and using the other global constraints. For example, the negation of `C: allDifferent(all(i in R) x[i])` is `or(ordered i,j in R) x[i] = x[j]` as `C` rewrites to `forall(ordered i,j in R) x[i] != x[j]`, and the negation of `forbiddenAssignments` is simply `allowedAssignments`.

Table 1. Syntax of OPL expressions handled by CPTEST

<i>Ctrs</i> ::=	<i>Ctr</i> <i>Ctrs</i>
<i>Ctr</i> ::=	<i>rel</i> <code>forall</code> (<i>rel</i>) <i>Ctrs</i> <code>or</code> (<i>rel</i>) <i>Ctrs</i> <code>if</code> (<i>rel</i>) <i>Ctrs</i> <code>else</code> <i>Ctrs</i> <code>allDifferent</code> (<i>rel</i>) <code>allMinDistance</code> (<i>rel</i>) <code>inverse</code> (<i>rel</i>) <code>forbiddenAssignments</code> (<i>rel</i>) <code>allowedAssignments</code> (<i>rel</i>) <code>pack</code> (<i>rel</i>)

Table 2. Faulty versions of the Golomb Ruler

	constraints of P present in the CPUT
CPUT1	cc1, cc9
CPUT2	cc1, cc2, cc7, cc9
CPUT3	cc1, cc2, cc7, cc8, cc9
CPUT4	cc1, cc2, cc3, cc4, cc6, cc7, cc8, cc9, cc10

We implemented algorithm [1](#) in CPTEST with several improvements. In particular, by noticing that it is unnecessary to search for non-conformities on constraints that are included in both the CPUT and the Model-Oracle, we implemented a simple rewriting system to check equality modulo Associativity-Commutativity (\equiv_{AC}). The system implements the following rules:

$$\left\{ \begin{array}{l} x \circ y \rightarrow y \circ x, \quad (x \circ y) \circ z \rightarrow x \circ (y \circ z), \quad x + 0 \rightarrow x, \\ x * 1 \rightarrow x, \quad x * 0 \rightarrow 0, \quad x \times (y \bullet z) \rightarrow (x \times y) \bullet (x \times z), \\ x < y \leftrightarrow y > x, \quad x \leq y \leftrightarrow y \geq x, \quad x - 0 \rightarrow x, \end{array} \right\}$$

where $\circ \in \{+, *, \wedge, \vee\}$, $\times \in \{*, \wedge, \vee\}$ and $\bullet \in \{+, \wedge, \vee\}$. In algorithm [1](#), the constraint C_i is discarded whenever there exists C'_i in D such as $C'_i \equiv_{AC} (C_i)$.

In addition, practical solutions for the handling of local variables and the computation of constraint projection exist: (a) Annotating the CPUT with constraints that define local variables ; (b) Computing constraint projection with Fourier’s elimination in the case of linear constraints ; (c) Eliminating false alarms with constraint checking. In CPTEST, we implemented (a) and (c).

The goal of our experimental evaluation was to check that CPTEST is able to detect faults in OPL programs. We fed CPTEST with faulty models coming from initial constraint program development. Indeed, we developed optimized models of two well-known CP problems, namely the Golomb rulers and the car sequencing problem, and we kept first versions of these models for which faults were found.

5.2 The Golomb Ruler Problem

The model-oracle of the Golomb rulers is given in part A of Fig [1](#) while part B contains a conform version of an optimized version of the model when the comment on constraint cc5 is removed. Let us call P this version. The four intermediate versions of the Golomb rulers we kept from our initial program development contain realistic faults, not invented for the experiment. Tab [2](#) shows the four faulty versions expressed with the constraints of P. Note that constraint cc6 breaks symmetries in the problem and then it removes solutions (valid Golomb rulers) w.r.t. the model-oracle. Constraint cc10 is not documented in P, it corresponds to forall(i in m..3*m) count(all(j in indexes)d[j],i)=1. For each CPUT, we studied its conformity w.r.t. the model-oracle (part A) using the four conformity relations. The results we got for an instance parameter $m = 8$ are given in Tab [3](#). For the *conf_bounds* relation, the interval [50, 100] was used to feed the relation, knowing that the global minimum is $x_m = 34$ when $m = 8$. Each

Table 3. Non-conformities found by CPTEST in various CPUTs of the Golomb rulers problem (timeout = 5 400s)

$m = 8$	<i>confone</i>	<i>confall</i>	<i>confbounds</i>	<i>confbest</i>	
Non-conf points	[0 7 8 18 24 26 35 44]	[17 18 20 25 34 45 49 55]	[0 2 3 6 11 58 72 86]	[0 1 3 6 10 15 24 33]	
CPUT1	T(s)	4.29s	21.45s	5.64s	7.31s
Non-conf points	[0 4 5 26 28 31 47 63]	[17 18 20 25 34 45 49 55]	[0 18 39 43 45 46 55 64]	[0 3 4 9 13 15 24 33]	
CPUT2	T(s)	5.62s	40.78s	4.64s	174.43s
Non-conf points	[0 4 5 26 28 31 47 63]	[0 4 5 26 28 31 47 63]	[0 18 39 43 45 46 55 64]	[0 3 4 9 13 15 24 33]	
CPUT3	T(s)	9.53s	45.78s	7.15s	389.04s
Non-conf points	[0 12 18 20 29 33 34 39]	[1 2 10 22 33 55 57 60]	[0 21 30 32 42 45 46 50]	[0 6 13 21 22 25 27 32]	
CPUT4	T(s)	12.60s	0.15s	9.01s	12.53s
Non-conf points	conf	[0 7 9 12 37 54 58 64]	conf	—	
P	T(s)	3 448.46s	0.18s	3 658.13s	timeout

time a non-confirmity was found, it was reported with the CPU time required to find it. Firstly, the four faulty CPUT were reported as being non-conforms and the time required for finding these non-conformities is acceptable (less than a few minutes in the worst case). Secondly, this experiment shows that the most practical conformance relations (i.e., *confone* and *confbounds*) are preferable to the other ones for efficiency reason. Indeed, for the first three CPUT, these relations gave results less than 10sec. Note that non-conformities are represented either by invalid Golomb rulers (e.g., $44 - 35 = 35 - 26 = 9$ in the CPUT1/*confone* case) or by valid Golomb rulers (e.g., CPUT1/*confall* case). In fact, a valid Golomb ruler r can be produced when the model-oracle is satisfied by r while the CPUT is refuted by r . These non-conformities correspond to cases where the CPUT misses solutions of the problem. Interestingly, P is shown as being non-conform with the *confAll* relation and the non-conformity that is found represent a valid Golomb ruler (i.e., [0 7 9 12 37 54 58 64]). In fact, recalling that P includes constraints that break the symmetries, this result was expected. Finally, note that conformity of P when *confbest* is selected was impossible to assess within the allocated time (timeout=5 400s). In fact, computing the global minimum of the Golomb ruler rapidly becomes hard even for small values of m (e.g., CPUT3/*confbest*).

Our experimental evaluation also had the goal to check that computing non-conformities with CPTEST was less difficult than computing solutions. For that, Fig. 6 shows: A) the CPU time required to find a global optimum for instances of the Golomb rulers (square points) and B) the CPU time required to find non-conformities with CPTEST with the *confbounds* conformity relation (lozenge points). The search heuristic used in both cases is the default heuristic of OPL, i.e. depth-first search with restarts, and branch-and-bound for the global optimization problem. CPTEST can find non-conformities when $m < 22$ in a reasonable amount of time because the hard global optimization problem has been relaxed in a simpler satisfaction problem, in order to deal with larger instances. This is the essence of the *confbounds* conformity relation.

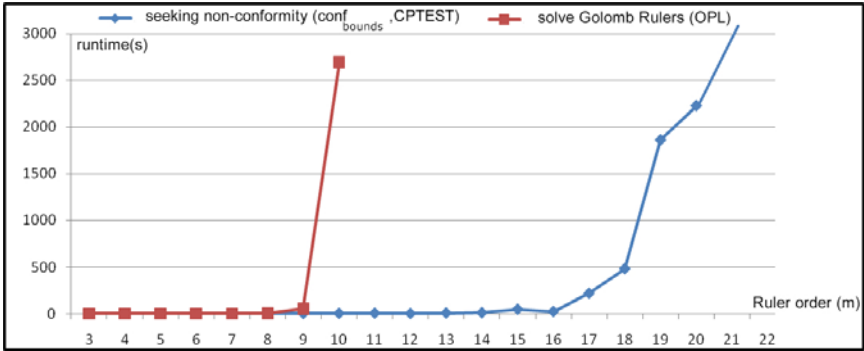


Fig. 6. Testing time and solving time comparison on the Golomb rulers

5.3 The Car Sequencing Problem

The car sequencing problem (CSeq) illustrates interesting features of CP including wide parameter settings, redundant, surrogate and global constraints addition, and specialized data structures definition. This is a constraint satisfaction problem that amounts to find an assignment of cars to the slots of a car-production company, which satisfies capacity constraints.

As a model-oracle of this problem, we took the model given in the OPL book [10]. In this model, capacity constraints are formalized by using constraints `r out of s`, saying that from each sub-sequence of `s` cars, a unit can produce at most `r` cars with a given option. Starting from this model, we built an optimized model by introducing several refinements, including a new data structure `setup[o,s]` which takes value 1 if option `o` is installed on slot `s`, redundant and global constraint addition (e.g., `pack` constraint). When building our improved model of car sequencing, we recorded four faulty constraint models that are used for experiments. Here again, the idea was to keep models that represent realistic faults instead of a posteriori injected faults. These four models are available online on the site mentioned above.

Tab. 4 gives the results of CPTTEST on two instances of the problem: an assembly line of 10 cars, 6 classes and 5 options ; an assembly line with 55 cars, 7 classes and 5 options. Using `confone`, CPTTEST reports non-conformities for the three first CPUT in less than 1sec for both instances. CPUT4 has no solution as the fault introduced on the `pack` constraint prunes dramatically the search space. This case is interesting as detecting this fault is really difficult. With the `confall` relation, the results are balanced as three instances were not detected as non-conformant within the allocated time slot. For example, in CPUT2, the capacity constraint of the first option is violated (1 out of 2). This fault results from a bad formulation but it is quickly detected with `confone`. When `confall` is selected, more constraints have to be negated and then our algorithm has to backtrack a lot, which explains the failure. The non-conformity reached in this case satisfies the model-oracle and violates CPUT2, so it represents a correct

Table 4. Non-conformities found by CPTEST in various CPUs of the car sequencing problem (timeout = 5 400s)

		<i>Conf_{one}</i>		<i>Conf_{all}</i>	
		10 slots	55 slots	10 slots	55 slots
CPU1	Non-conf points	4 5 3 6 4 6 5 1 3 2	p1	4 5 4 6 3 6 5 1 3 2	—
	T(s)	0.30s	1.23s	2.49s	timeout
CPU2	Non-conf points	4 6 3 1 5 2 3 5 4 6	p2	5 4 3 5 4 6 2 6 3 1	—
	T(s)	0.85s	1.65s	1.20s	timeout
CPU3	Non-conf points	5 2 3 6 1 4 3 6 4 5	p3	5 4 3 5 4 6 2 6 3 1	—
	T(s)	0.24s	0.70s	90.73s	timeout
CPU4	Non-conf points	conf	conf	1 3 6 2 6 4 5 3 4 5	p4
	T(s)	0.96s	1.06s	1.26s	100.22s
P	Non-conf points	conf	—	6 4 5 3 4 5 2 6 3 1	—
	T(s)	3.01s	timeout	0.17s	timeout
p1 = 6 5 6 4 5 2 4 4 4 3 5 6 7 6 3 3 3 5 6 4 5 5 2 2 7 3 4 2 5 5 4 1 3 4 1 6 4 3 1 5 3 3 6 1 6 7 7 2 6 3 1 6 4					
p2 = 7 1 6 3 4 6 1 7 3 2 5 1 7 3 5 4 2 6 6 6 4 3 6 5 3 4 4 2 4 6 1 3 7 5 5 2 5 5 3 7 6 3 1 6 4 3 5 4 2 4 6 5 5 4 3					
p3 = 4 3 1 5 6 5 5 1 2 4 2 3 6 6 6 3 2 5 2 1 7 4 4 4 3 3 3 5 4 3 6 4 6 6 4 1 7 3 1 5 6 4 2 5 7 6 3 5 5 6 7 4 3 7 5					
p4 = 1 3 6 2 5 4 3 5 2 6 4 5 3 4 5 2 6 3 5 4 4 5 3 7 6 4 1 3 6 7 1 7 6 3 1 4 6 7 5 2 6 3 1 7 6 4 5 4 3 5 4 6 2 5 3					

assembly line that CPU2 excludes from its solutions. Therefore, we can conclude that CPU2 adds and removes solutions which make it difficult to detect as non-conform.

6 Conclusion

In this paper, we introduced for the first time a testing framework that is adapted to standard CP development processes. The framework is built on solid notions such as conformity relations, oracles and test purposes that are specific to CP. We also presented CPTEST an implementation of our framework dedicated to the testing of OPL programs and evaluated it on difficult instances of two well-known constraint problems, namely the Golomb ruler and car-sequencing problem. Our experimental evaluation shows that CPTEST can efficiently detect non-trivial faults in faulty versions of those two problems. A desirable extension of our framework and tool concerns its application to other more open CP platforms. In particular, we would like to apply our conformity relations, oracles and testing notions to GECODE or CHOCO programs as we could intervene on the core constraint solver of these systems. Developing notions of test coverage similar of those that can be found in conventional programming requires instrumenting the solver, something that was just not possible with the black-box solver of OPL.

Acknowledgment

We are very grateful to Olivier Lhomme who pointed us the problem of out-of-scope variables. Many thanks also to Michel Rueher, Laurent Granvilliers and Nicolas Beldiceanu for helpful comments on early presentations.

References

1. Collavizza, H., Rueher, M., Van Hentenryck, P.: Cpbpv: A constraint-programming framework for bounded program verification. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 327–341. Springer, Heidelberg (2008)
2. Deransart, P., Maluszyński, J. (eds.): DiSCiPl 1999. LNCS, vol. 1870. Springer, Heidelberg (2000)
3. Flener, P., Pearson, J., Agren, M.: Garcia-Avello C., M. Celiktin, and S. Dissing. Air-traffic complexity resolution in multi-sector planning. *Journal of Air Transport Management* 13(6), 323–328 (2007)
4. Gotlieb, A.: Tcas software verification using constraint programming. *The Knowledge Engineering Review* (2009) (accepted for publication)
5. Holland, A., O’Sullivan, B.: Robust solutions for combinatorial auctions. In: ACM Conference on Electronic Commerce (EC-2005), pp. 183–192 (2005)
6. Junker, U., Vidal, D.: Air traffic flow management with ilog cp optimizer. In: International Workshop on Constraint Programming for Air Traffic Control and Management, 7th EuroControl Innovative Research Workshop and Exhibition, INO 2008 (2008)
7. Langevine, L., Deransart, P., Ducassé, M., Jahier, E.: Prototyping clp(fd) tracers: a trace model and an experimental validation environment. In: WLPE (2001)
8. Rankin, W.T.: Optimal golomb rulers: An exhaustive parallel search implementation. Master’s thesis, Duke University, Durham (1993)
9. Sahinidis, N.V., Twarmalani, M.: Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming. Kluwer Academic Publishers, Dordrecht (2002)
10. Van Hentenryck, P.: The OPL optimization programming language. MIT Press, Cambridge (1999)
11. Weyuker, E.J.: On testing non-testable programs. *Computer Journal* 25(4), 465–470 (1982)
12. Zhu, H., Hall, P.A.V., May, J.H.R.: Software unit test coverage and adequacy. *ACM Comput. Surv.* 29(4), 366–427 (1997)

On the Containment of Forbidden Patterns Problems

Florent Madelaine

LIMOS, UMR CNRS 6158, Campus des Cézeaux,
IUT informatique, B.P. 86, 63172 AUBIERE, France
fmadelaine@u-clermont1.fr

Abstract. Forbidden patterns problems are a generalisation of (finite) constraint satisfaction problems which are definable in Feder and Vardi's logic MMSNP [1]. In fact, they are examples of infinite constraint satisfaction problems with nice model theoretic properties introduced by Bodirsky [2]. In previous work [3], we introduced a normal form for these forbidden patterns problems which allowed us to provide an effective characterisation of when a problem is a finite or infinite constraint satisfaction problem. One of the central concepts of this normal form is that of a *recolouring*. In the presence of a recolouring from a forbidden patterns problem Ω_1 to another forbidden patterns problem Ω_2 , containment of Ω_1 in Ω_2 follows. The converse does not hold in general and it remained open whether it did in the case of problems being given in our normal form. In this paper, we prove that this is indeed the case. We also show that the recolouring problem is Π_2^P -hard and in Σ_3^P .

Keywords: Constraint Satisfaction, Graph Homomorphism, Logic in Computer Science, Monadic Second Order Logic, Computational Complexity.

1 Introduction

Feder and Vardi [1] conjectured nearly 20 years ago that the class of non-uniform constraint satisfaction problems (CSP) has a dichotomy, that is that every problem in this class is either tractable or NP-complete. In contrast, it is believed that NP does not have the dichotomy property, as by Ladner's theorem [4], if $P \neq NP$, then there are problems in NP which are neither in P nor NP-complete. The dichotomy conjecture remains open though progress has been made using the central notion of polymorphisms in the mid nineties by Cohen, Jeavons and others and at the turn of the century great progress followed from Bulatov's powerful algebraic approach involving tame congruence theory (see [5] for a recent survey).

Descriptive complexity theory seeks to classify problems, *i.e.*, classes of finite structures, as to whether they can be defined using formulae of some specific logic, in relation to their computational complexity. One of the seminal results in descriptive complexity is Fagin's theorem [6] which states that a problem

can be defined in existential second-order logic (ESO) if, and only if, it is in the complexity class NP. In their influential paper [1], Feder and Vardi also introduced the logic MMSNP, a syntactic fragment of monotone monadic ESO which is intimately linked to CSP. It is thought to be the largest such fragment to exhibit a dichotomy¹ and the derandomisation by Kun [7] of a lemma used by Feder and Vardi implies that MMSNP exhibits a dichotomy if, and only if, the dichotomy conjecture for CSP holds.

The logic MMSNP does not capture CSP: every problem in CSP can be defined in MMSNP but there are problems in MMSNP which are not in CSP [18]. In previous work with Iain Stewart [9,3], we provided an effective method to decide given a sentence of MMSNP whether it defines a problem in CSP or not. It turns out that these problems in MMSNP that are not in CSP are actually *constraint satisfaction problems with an infinite domain*, whose templates have nice model theoretic properties, introduced by Bodirsky [10]. So our previous result provides in fact a decision procedure that can tell whether a sentence of MMSNP defines a finite or an infinite CSP problem. In contrast, when the input of a problem definable in MMSNP is restricted to be of bounded degree, or from a proper minor closed class or more generally of bounded expansion, the restricted problem becomes a restricted *finite* CSP [11]. It is important to note that though there are infinite CSP *à la* Bodirsky which are not definable in MMSNP, this logic defines a large infinite class of natural infinite CSP which are worth studying in their own rights. For example, the complexity of problems in MMSNP have recently been investigated in the special case of monochromatic and loopless forbidden patterns [12].

Combinatorially, a problem in CSP can be seen as a homomorphism problem represented by a finite structure \mathcal{T} , the so-called template. It is well known that the containment of CSP corresponds exactly to the existence of a homomorphism from one template to another. More precisely, the CSP with template \mathcal{T}_1 is contained in the CSP with template \mathcal{T}_2 if, and only if, there is a homomorphism from \mathcal{T}_1 to \mathcal{T}_2 . Therefore the category of relational structures and homomorphisms crops up naturally in the study of CSP [13].

Combinatorially, a problem in MMSNP can be represented by a finite set of coloured obstructions, the so-called *forbidden patterns*, and an instance is accepted if, and only if, it can be coloured while avoiding the presence of these patterns. The key ingredient of our previous result was to refine Feder and Vardi's normal form of MMSNP to take into account the fact that some colours might actually be redundant in the representation of the problem. To formalise this, we introduced the notion of a recolouring from a forbidden patterns problem Ω_1 to another forbidden patterns problem Ω_2 and showed that in the presence of such a recolouring, containment of the problem Ω_1 in the problem Ω_2 followed. The converse does not hold in general and it remained open whether it did in the case of problems being given in our normal form. In this paper, we prove that this is indeed the case. It follows that representations of forbidden patterns problems given in a normal form and recolourings provide us with the right category in

¹ Feder and Vardi showed that monotone monadic SNP with \neq does not have a dichotomy.

the context of MMSNP. It would be interesting to settle the complexity of the containment of forbidden patterns problems. We investigate as a first step the complexity of the recolouring problem and show that it is in Σ_3^P and that it is Π_2^P -hard.

This paper is organised as follows. In the next section, as the reader may not be familiar with MMSNP, we shall introduce key concepts informally, mostly by discussing examples, prove simple cases of our main result to illustrate our method, and finally state our main result. In Section 3, we detail and adapt the computational equivalence between CSP and MMSNP given by Feder and Vardi. In Section 4, we prove our main result. We conclude with a discussion of the complexity of some related problems.

2 Preliminaries

Existential Second Order Logic. Fagin’s theorem equates definability in ESO with membership in the complexity class NP. For example, the class of 3-colourable graphs can be defined using a sentence of the following form.

$$\begin{aligned} \Phi_1 := \exists R, G, B, \text{ three sets partitioning the vertices} \\ \forall x, y, \neg(E(x, y) \wedge R(x) \wedge R(y)) \wedge \neg(E(x, y) \wedge G(x) \wedge G(y)) \\ \wedge \neg(E(x, y) \wedge B(x) \wedge B(y)) \end{aligned}$$

A graph is represented as a relational structure whose domain consists of vertices equipped with a single binary predicate E representing the edge relation. The above sentence has two kinds of quantifiers: second-order variables (always upper-case) are interpreted as relations, like R which is interpreted as a set of vertices, and first-order variables (always lower case), like x , which is interpreted as a vertex. The three second order predicates R, G and B stand for three colours, say red, green and blue and the sentence asserts that the vertices may be coloured with these three colours in such a way that for every edge in the graph, the extremities have different colours.

In this paper, we shall only need second-order predicates that are sets, the so-called *monadic* predicates, and we shall only allow them to be existentially quantified as in the above example. Note that finitely many sets of vertices correspond essentially to a partition of the vertices in distinct *colours*. In combinatorial terms this means that in order to check a property we have to guess some colours for each vertex before verifying some first-order property over the coloured graph. Let us clarify this with another example.

$$\begin{aligned} \Phi_2 := \exists M, N \forall x, y, \neg(\neg M(x) \wedge \neg N(x)) \\ \wedge \neg(E(x, y) \wedge M(x) \wedge N(x) \wedge M(y) \wedge N(y)) \\ \wedge \neg(E(x, y) \wedge \neg M(x) \wedge N(x) \wedge \neg M(y) \wedge N(y)) \\ \wedge \neg(E(x, y) \wedge M(x) \wedge \neg N(x) \wedge M(y) \wedge \neg N(y)) \end{aligned}$$

There are two monadic predicates M and N in Φ_2 and for a given vertex x there are four cases to consider: x is in both M and N ($M(x) \wedge N(x)$ holds), x is in M but not in N ($M(x) \wedge \neg N(x)$ holds) etc. So the above sentence disallows one of the colour (with the conjunct $\neg(\neg M(x) \wedge \neg N(x))$) and states for the three other colours that an edge can not have both extremities of the same colour. In other words, this sentence defines also the fact that a graph is 3-colourable.

Monotone Monadic Strict NP without inequalities. The two sentences Φ_1 and Φ_2 have a particular syntactic form: \exists monadic predicates, \forall variables ranging over vertices, followed by a quantifier-free first-order formula. Such sentences form the fragment SNP of ESO. It turns out that many combinatorial problems are definable in SNP, in particular every problem in CSP can be defined by a SNP sentence. For example, in the case of 3-colourability, we may use the sentence Φ_2 . Let us explain in a bit more detail how we may build this sentence in a systematic fashion. Recall first that for a CSP with template \mathcal{J} , a structure \mathcal{A} is a yes-instance if, and only if, there exists a *homomorphism* from \mathcal{A} to \mathcal{J} . That is, a mapping h from the domain of \mathcal{A} to that of \mathcal{J} such that every arc in \mathcal{A} is mapped to an arc in \mathcal{J} (assuming we deal with digraphs for now for the sake of simplicity). The 3-colourability problem, recast as a digraph problem, has as template \mathcal{J} the digraph with 3 vertices and all possible arcs that are not self-loops. Viewing the 3 elements of \mathcal{J} as colours, we have readily explained how to use 2 monadic predicates M and N and one forbidden combination of them $\neg(\neg M(x) \wedge \neg N(x))$ to encode three colours. In order to enforce a homomorphism, we now encode the non-arcs of \mathcal{J} by adding negated conjuncts, one for each non-arc. For example, if $M(x) \wedge N(x)$ stands for the colour corresponding to the first vertex of \mathcal{J} and since there is no self-loop around this vertex, we add the following negated conjunct to the sentence:

$$\neg(E(x, y) \wedge M(x) \wedge N(x) \wedge M(y) \wedge N(y)).$$

Doing this with every non-arc, we obtain the sentence Φ_2 given above. It is important to note that the sentence we build this way uses only *monadic* predicates. Furthermore, the first-order part is a conjunction of negated conjuncts; and, in every negated conjunct atoms from the input (the edge symbol E in our examples) appears always positively. This means that the sentence is *monotone*. Finally, we never use the symbol \neq . We have therefore built a sentence of SNP that is *monadic*, *monotone* and *without inequality*. The sentences of SNP satisfying these three restrictions form the logic MMSNP introduced by Feder and Vardi. As we may build such a sentence for every template, we now know that

$$\text{CSP} \subseteq \text{MMSNP}.$$

Some sentences of MMSNP give rise to problems that are not in CSP and are in fact constraint satisfaction problems with an infinite template. For example,

$$\Psi_1 := \forall x, \forall y, \forall z, \neg(E(x, y) \wedge E(y, z) \wedge E(z, x))$$

expresses that there are no oriented 3-cycles in a digraph (and also no self-loop as the variables may be equal). It is not difficult to see that this problem is not in CSP. Assume for contradiction that there exists a template \mathcal{T} with n elements for this problem. We may build a yes-instance \mathcal{A} for ψ_1 as follows: take $n + 1$ vertices and add between any pair of distinct vertices a directed path of length 3. By assumption, there exists a homomorphism from \mathcal{A} to \mathcal{T} . This homomorphism must identify two distinct elements joined by a directed 3-path. Hence, \mathcal{T} contains a loop or an oriented 3-cycle and is a no-instance which is absurd as the template is always a yes-instance.

The problem defined by Ψ_1 is in fact a CSP with an infinite template. It is not difficult to construct an infinite template for this problem: simply take as a template the disjoint union of its yes-instances². This infinite template is not particularly interesting, however, we may also construct for this problem an infinite template that has a nice model theoretic property called ω -categoricity. From now on, by infinite CSP, we mean a problem with such a nice template. This property means in particular that the Galois-connection used in the finite case can be successfully adapted and that some logico-algorithmic results such as those involving Datalog still hold. We will refrain from going into more details and refer to Bodirsky’s survey [2] on his pioneering work on infinite CSP.

Obstructions and containment. Note that the negated conjunct

$$\neg(E(x, y) \wedge E(y, z) \wedge E(z, x))$$

in Ψ_1 essentially forbids the occurrence of an oriented 3-cycle. However, since the variables x, y and z may take the same value, this means in fact that we forbid the existence of a homomorphism from the oriented 3-cycle to the instance. Hence, the problem defined by Ψ_1 can be seen as a dual problem to a CSP. Whereas in the case of CSP we ask whether there is a homomorphism from the instance \mathcal{A} to the template \mathcal{T} , we will ask here whether there is no homomorphism from an obstruction \mathcal{F} to the instance \mathcal{A} . In the case of more than one obstruction, we have essentially the fragment of MMSNP that has no monadic predicate (sentences of MMSNP that are also first-order). In general, such a problem is known to be an infinite CSP [14]. Let us consider two such problems Ω_1 and Ω_2 given by two sets of obstructions \mathcal{F}_1 and \mathcal{F}_2 . We will insist for simplicity for the obstructions to be connected³. We say that the problem Ω_1 is *contained* in Ω_2 if, and only if, for any instance \mathcal{A} , if \mathcal{A} is a yes-instance of Ω_1 then \mathcal{A} is a yes-instance of Ω_2 . When is Ω_1 contained in Ω_2 ? A simple criteria defined in terms of existence of homomorphisms between the obstructions characterises containment in this simple case.

Proposition 1 ([9], see also [15]). Ω_1 is contained in Ω_2 if, and only if, for every obstruction \mathcal{F}_2 in \mathcal{F}_2 there exists an obstruction \mathcal{F}_1 in \mathcal{F}_1 such that there is a homomorphism from \mathcal{F}_1 to \mathcal{F}_2 .

² This is true in general for any monotone problem that is closed under disjoint union.

³ This is not a strong hypothesis as a problem with a disconnected obstruction is in fact the disjoint union of problems with connected obstructions.

The main result of this paper is a generalisation of the above result to the case where the obstructions are coloured, that is when the corresponding MMSNP sentences are no longer first-order sentences.

Forbidden patterns problems. Another example of a problem that is in MMSNP but not in CSP is:

$$\Psi_2 := \exists M, \forall x, y, z, \neg(E(x, y) \wedge E(y, z) \wedge E(z, x) \wedge M(x) \wedge M(y) \wedge M(z)) \\ \wedge \neg(E(x, y) \wedge E(y, z) \wedge E(z, x) \wedge \neg M(x) \wedge \neg M(y) \wedge \neg M(z)).$$

We have a single monadic predicate which encodes two colours, say white and black. The two negated conjuncts forbid two vertex-coloured structures, namely a white oriented 3-cycle \mathcal{F}'_1 and a black oriented 3-cycle \mathcal{F}'_2 . Thus, the problem defined by Ψ_2 accepts an instance \mathcal{A} whenever its vertices can be coloured in white and black into a structure \mathcal{A}' such that there is neither a homomorphism from \mathcal{F}'_1 to \mathcal{A}' nor a homomorphism from \mathcal{F}'_2 to \mathcal{A}' .

In general a *forbidden patterns problem* Ω is given by a finite set of coloured structures. We insist that each structure is *connected* and *contains at least one tuple*. It makes sense to formalise the (vertex-)colouring of a structure by a homomorphism into some structure describing the colours. So Ω is given by a structure \mathcal{T} representing the colours and a set \mathcal{F}' of \mathcal{T} -coloured structures, the so-called *forbidden patterns*.

A \mathcal{T} -coloured structure is a pair (\mathcal{F}, f) where f is a homomorphism from \mathcal{F} to \mathcal{T} which describes the colouring. The notion of structure homomorphism generalises naturally to coloured structures: given two \mathcal{T} -coloured structures (\mathcal{F}, f) and (\mathcal{G}, g) , a homomorphism h from (\mathcal{F}, f) to (\mathcal{G}, g) is simply a homomorphism from \mathcal{F} to \mathcal{G} that preserves the colours, that is such that $f = g \circ h$.

An instance \mathcal{A} of the problem Ω is a *yes-instance* if, and only if, there exists a homomorphism h from \mathcal{A} to \mathcal{T} such that there is no homomorphism from any forbidden pattern (\mathcal{F}, f) in \mathcal{F}' to (\mathcal{A}, h) .

When h is not a homomorphism or when there is a homomorphism from some forbidden pattern (\mathcal{F}, f) in \mathcal{F}' to (\mathcal{A}, h) , we say that (\mathcal{A}, h) is *not valid* w.r.t. Ω . We denote by FPP the class of forbidden patterns problems. Forbidden patterns problems are known to be infinite CSP [10] and every sentence in MMSNP captures a finite union of problems in FPP [3].

Recolouring. A recolouring is a homomorphism which states how the colours of a problem Ω_1 can be transformed into colours of a problem Ω_2 . Let us recall the formal definition before looking at an example.

Definition 2 (recolouring [9,3]). *Let Ω_1 (respectively, Ω_2) be a forbidden patterns problem given by \mathcal{T}_1 and a set \mathcal{F}'_1 of \mathcal{T}_1 -coloured forbidden patterns (respectively, \mathcal{T}_2 and \mathcal{F}'_2).*

A recolouring from Ω_1 to Ω_2 is a homomorphism r from \mathcal{T}_1 to \mathcal{T}_2 such that for every (\mathcal{F}_2, f_2) forbidden by Ω_2 , any of its inverse image (\mathcal{F}_2, f_1) under r is not valid w.r.t. Ω_1 . In other words, for every \mathcal{T}_2 -coloured pattern (\mathcal{F}_2, f_2) in \mathcal{F}'_2 ,

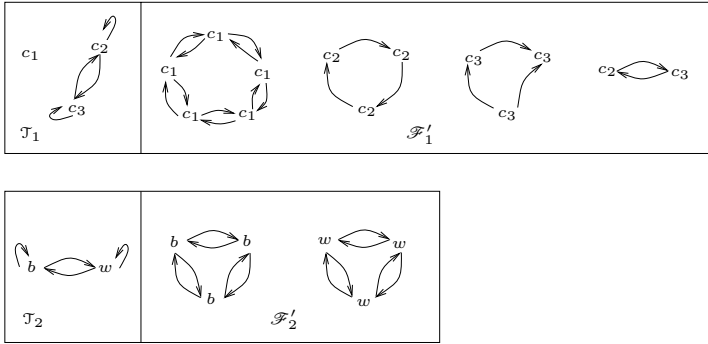


Fig. 1. Two forbidden patterns problems

and for any \mathcal{T}_1 -coloured structure (\mathcal{F}_2, f_1) such that $f_2 = r \circ f_1$, there exists a forbidden pattern (\mathcal{G}_1, g_1) in \mathcal{F}'_1 and a homomorphism h from (\mathcal{G}_1, g_1) to (\mathcal{F}_2, f_1) .

Note how this definition generalises the condition between the obstructions given in Proposition 1. We already know that the existence of a recolouring implies containment.

Proposition 3 ([9,3]). *If there is a recolouring r from a forbidden patterns problem Ω_1 to a forbidden patterns problem Ω_2 then Ω_1 is contained in Ω_2 .*

Example 4. We consider the two forbidden patterns given on Figure 1 (note how the colours of the vertices of a forbidden pattern are simply given by labelling a vertex with its colour). The problem represented by \mathcal{T}_2 and \mathcal{F}'_2 is a variant of the problem defined by Ψ_2 in which triangles have arcs in both directions. Let r be the mapping from the colours of the first problem, namely $\{c_1, c_2, c_3\}$ to those of the second problem, namely $\{b, w\}$, that maps c_1, c_2 and c_3 to b . Note that r is indeed a homomorphism from \mathcal{T}_1 to \mathcal{T}_2 . The only forbidden pattern of the second problem whose colours are in the image of r is the black triangle (the first forbidden pattern of \mathcal{F}'_2 listed on the figure). We need to show that every triangle whose vertices is coloured via r^{-1} are invalidated by the first problem. This can happen in two ways: the colouring may not be a homomorphism to \mathcal{T}_1 , or some forbidden pattern in \mathcal{F}'_1 invalidates it. If the colours of the three vertices of the triangle are replaced by c_1 , then the 5-cycle (the first forbidden pattern of \mathcal{F}'_1 listed on the figure) invalidates this choice of colours. Similarly, if the vertices are all coloured by c_2 only or c_3 only then the two next forbidden patterns on the figure invalidate these choices. If the colours of the three vertices of the triangle are replaced by c_1 and other colours then the colouring is not a homomorphism to \mathcal{T}_1 . If the colours are replaced by c_2 and c_3 but not c_1 then the last forbidden pattern listed on the figure invalidates this choice. This shows that r is a recolouring from the first problem to the second problem.

Normal Form for Forbidden Patterns Problems. In this paper, we prove that the converse of Proposition 3 holds, when the two problems are given in the *normal form*. Note that this can always be done.

Theorem 5 (3). *Every forbidden patterns problem can be given by a representation in the normal form.*

We shall recall shortly what conditions this normal form entails. Let us first introduce some vocabulary. We say that a coloured structure is *weakly valid* w.r.t. a forbidden patterns problem if there is no *injective* homomorphism from a forbidden pattern into it. A forbidden patterns that consists of a coloured structure with a single tuple that mentions each element exactly once⁴ is said to be *conform*. When a forbidden pattern is conform, we may drop it from the list of forbidden patterns and enforce its constraint by amending the structure \mathcal{T} accordingly (by removing the corresponding tuple from \mathcal{T}). A forbidden patterns problem Ω is given by a structure \mathcal{T} and a set of forbidden \mathcal{T} -coloured structures \mathcal{F}' . The pair $(\mathcal{T}, \mathcal{F}')$ is called a *representation* of Ω . If every recolouring from $(\mathcal{T}, \mathcal{F}')$ to itself is an automorphism of \mathcal{T} then we say that the representation $(\mathcal{T}, \mathcal{F}')$ is a *core*.

Definition 6 (Normal Form 3). *A representation $(\mathcal{T}, \mathcal{F}')$ of a forbidden patterns problem Ω is said to be in the normal form if, and only if it satisfies the following six conditions.*

- (p1). *An instance is valid if, and only if, it is weakly valid.*
- (p2). *Every pattern of \mathcal{F}' is a core (as a coloured structure).*
- (p3). *It is not the case that (\mathcal{F}_1, f_1) is a substructure of (\mathcal{F}_2, f_2) , for any distinct patterns (\mathcal{F}_1, f_1) and (\mathcal{F}_2, f_2) in \mathcal{F}' .*
- (p4). *No pattern of \mathcal{F}' is conform.*
- (p5). *Every forbidden pattern is biconnected.*
- (p6). *The representation $(\mathcal{T}, \mathcal{F}')$ is a core.*

Example 7. Let Ω_4 be the problem given on the top of Figure 2. We shall discuss briefly how its normal form is computed without explaining why the obtained problem is equivalent, for further details please refer to 3.

First we enforce \mathbf{p}_1 to \mathbf{p}_3 simply by taking the homomorphic image of the forbidden pattern, keeping only the minimal ones with respect to injective homomorphisms. Note that \mathbf{p}_4 holds also in the representation of the problem we obtain this way which is given in the second row on the figure.

Next, we enforce \mathbf{p}_5 by splitting the path of length two along its articulation point and copying its colour c into two new colours b and w , one for the substructure to the left of this articulation point, one for the substructure to the right of this articulation point. Replacing elsewhere the colour c by w and b in all possible ways and simplifying again by keeping the minimal patterns to enforce \mathbf{p}_3 , we obtain the representation which is given in the fourth row of the figure. Note that it no longer satisfies \mathbf{p}_4 .

⁴ Self-loops and their generalisation like $R(x, x, y)$ are not conform.

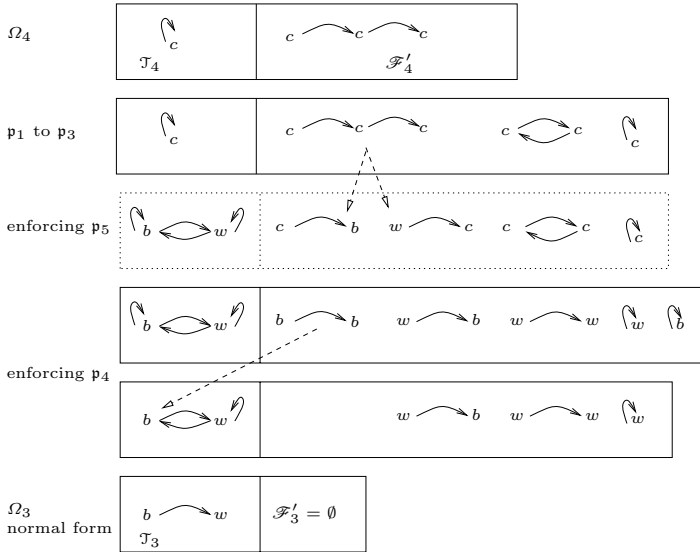


Fig. 2. Computing the normal form

We enforce progressively p_4 by removing the conform forbidden patterns and removing the corresponding tuple in the structure describing the colours. We also remove any forbidden pattern that is no longer a coloured structure. We finally obtain this way the problem Ω_3 given in the last row on the figure.

The mapping r which sends w and b to the single colour c of Ω_4 is a recolouring from Ω_3 .

Conversely, there is no recolouring from Ω_4 to Ω_3 as there is no homomorphism from \mathcal{T}_4 to \mathcal{T}_3 , since the former is a self-loop and the latter has no self-loop.

Note that the two problems Ω_3 and Ω_4 coincide and that Ω_3 is given in the normal form but that Ω_4 is not (its only forbidden pattern fails to be biconnected).

We are now ready to state the main result of this paper.

Theorem 8 (main result). *Let Ω_1 and Ω_2 be two forbidden patterns problems given in the normal form over the relational signature σ . Ω_1 is contained in Ω_2 if, and only if, there is a recolouring from Ω_1 to Ω_2 .*

Another case where it is not too hard to see that the converse of Proposition 1 holds is when Ω_2 is in CSP. Though this case is subsumed by our main result, its proof will serve as a good warm-up. In particular, it will allow us to introduce a key ingredient which is a generalisation by Feder and Vardi of a result due to Erdős.

Proposition 9. *Let Ω_1 and Ω_2 be two forbidden patterns problems. If both problems are given in the normal form and Ω_2 is in CSP then Ω_1 is included in Ω_2 if, and only if, there is a recolouring r from Ω_1 to Ω_2 .*

Recall that the *girth* of a structure is the length of its shortest cycle (and so if there are no cycles then the structure has infinite girth).

Lemma 10 (Erdős lemma [1]). *Fix two positive integers r and s . For every structure \mathcal{B} , there exists a structure \mathcal{D} such that: the girth of \mathcal{D} is greater than r ; there is a homomorphism from \mathcal{D} to \mathcal{B} ; and for every structure \mathcal{C} of size at most s , there is a homomorphism from \mathcal{B} to \mathcal{C} if, and only if, there is a homomorphism from \mathcal{D} to \mathcal{C} .*

Proof (of Proposition 9). As Ω_2 is in the normal form and in CSP, this means that that $\mathcal{F}'_2 = \emptyset$ [3]. Thus, in this case a recolouring is nothing other than a homomorphism from \mathcal{T}_1 to \mathcal{T}_2 . In particular if \mathcal{T}_1 is a yes-instance of Ω_1 then we are done. However, this is in fact not true in general.

By assumption Ω_1 is given in the normal form. This means that \mathcal{T}_1 is a no-instance of Ω_1 unless $\mathcal{F}'_1 = \emptyset$ [3]. We use Erdős Lemma: we choose r greater than the largest forbidden patterns in \mathcal{F}'_1 ; s to be $|\mathcal{T}_2|$, the size of \mathcal{T}_2 ; and $\mathcal{B} := \mathcal{T}_1$.

We claim that the structure \mathcal{D} obtained from the lemma in this way is in fact a yes-instance of Ω_1 . This is because the homomorphism, say d_1 , given by the lemma from \mathcal{D} to $\mathcal{B} = \mathcal{T}_1$ gives us a valid colouring w.r.t. Ω_1 . To see this, we use the fact that Ω_1 is given in the normal form: it suffices to show that (\mathcal{D}, d_1) is weakly valid; and, for every forbidden pattern (\mathcal{F}_1, f_1) , the structure \mathcal{F}_1 is biconnected and must contain a cycle, so it can not occur as a substructure of \mathcal{D} which has a girth greater than the size of any forbidden patterns.

By containment of Ω_1 in Ω_2 it follows that \mathcal{D} is a yes-instance of Ω_2 and that there is a homomorphism from \mathcal{D} to \mathcal{T}_2 . Hence, by construction of \mathcal{D} this means that there is a homomorphism from $\mathcal{B} = \mathcal{T}_1$ to \mathcal{T}_2 and that we are done. \square

3 From Forbidden Patterns Problem to CSP and Back

The following result is an adaptation of the ideas of Feder and Vardi’s reduction of MMSNP to CSP [1] to forbidden patterns problems. We shall only sketch the proof. A detailed proof using the same notation is available in [9]. There is a small difference here, as the signature of the CSP is now parameterised by a set of patterns that must include the patterns from the forbidden patterns problem considered but may include more. This result is one of the ingredient of the proof of our main result. We denote by $\text{CSP}(-, \mathcal{T})$ the (non-uniform) constraint satisfaction problem with template \mathcal{T} and by $\text{CSP}(\text{girth} > \gamma, \mathcal{T})$ its restriction to input of girth greater than γ .

Theorem 11. *Let Ω be a forbidden patterns problem given in the normal form over the relational signature σ . Let \mathcal{F} be a set of biconnected σ -structures that includes all structures involved in patterns forbidden by Ω . Let γ be a fixed integer greater than the largest structure in \mathcal{F} .*

There exists a relational signature τ , a τ -structure \mathcal{T}_Ω , and two first-order interpretations Π and Π^{-1} such that:

- τ extends σ with new symbols, one symbol $R_{\mathcal{F}}$ of arity $|\mathcal{F}|$ for each \mathcal{F} in \mathcal{F} ;

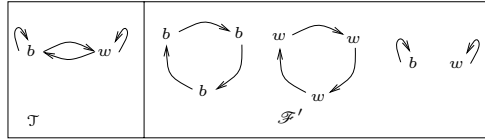


Fig. 3. No-Monochromatic-Triangle

- Π is a quantifier-free first-order interpretation using conjunction only;
- Π^{-1} is a first-order interpretation;
- $\Pi^{-1} \circ \Pi$ is the identity over σ -structures;
- Ω reduces to $\text{CSP}(-, \mathcal{T}_\Omega)$ via Π ; and,
- $\text{CSP}(\text{girth} > \gamma, \mathcal{T}_\Omega)$ reduces to Ω via Π^{-1} .

We sketch the proof of this result in the remaining of this section, providing an example to help the reader understand the main ideas⁵.

Example 12. We consider the forbidden patterns problem defined by the sentence Ψ_2 in the introduction. It is a variant of the well-known NP-complete problem **No-Monochromatic-Triangle**. It is given in its normal form on Figure 3. The signature of this problem is $\sigma = \langle E \rangle$ where E is binary which we extend to a *new signature* $\tau = \langle E, R, S \rangle$ where R is ternary and S unary (R encodes the 3-cycles and S the self-loops). The *interpretation* Π from σ to τ is given by: $\varphi_R(y_1, y_2, y_3) := E(y_1, y_2) \wedge E(y_2, y_3) \wedge E(y_3, y_1)$, $\varphi_S(y_1) := E(y_1, y_1)$ and $\varphi_E(y_1, y_2) := E(y_1, y_2)$. The *interpretation* Π^{-1} from τ to σ is given by the formula ψ_E which is as follows:

$$(E(y_1, y_2)) \vee (y_1 = y_2 \wedge S(y_1)) \vee (\exists x R(y_1, y_2, x) \vee R(x, y_1, y_2) \vee R(y_2, x, y_1)).$$

The structure \mathcal{T}_Ω has two elements b and w and, relations $E := \{b, w\}^2$, $S := \emptyset$ and $T := \{b, w\}^3 \setminus \{(b, b, b) (w, w, w)\}$. □

Signature of the CSP. The problem Ω is represented by a σ -structure \mathcal{T} and a list of forbidden \mathcal{T} -coloured structures $\{(\mathcal{F}_1, f_1), (\mathcal{F}_2, f_2), \dots, (\mathcal{F}_n, f_n)\}$. Let \mathcal{F} be the set of the σ -structures that consists of the structures \mathcal{F}_i considered up to isomorphism. For every \mathcal{F} in \mathcal{F} , we introduce a new symbol $R_{\mathcal{F}}$ of arity $|\mathcal{F}|$. Let τ be the signature that consists of the symbol of σ together with the new symbols $R_{\mathcal{F}}$.

Interpretation from the forbidden patterns problem to the CSP. Let $\varphi_{\mathcal{F}}$ be the quantifier-free part of the canonical conjunctive query of \mathcal{F} , that is:

$$\varphi_{\mathcal{F}} := \bigwedge_{R \in \sigma} \bigwedge_{R^{\mathcal{F}}(\bar{x}) \text{ holds}} R(\bar{x})$$

⁵ We advise the reader to go through the proof and progress in parallel on the example.

Let $\varphi_R := R(\bar{x})$. Let Π be the interpretation from σ to τ given by the formulae $\varphi_{\mathcal{F}}$ and the formulae φ_R . Note that Π is a quantifier-free interpretation of width one using only conjunction.

Interpretation from the CSP to the forbidden patterns problem. Let Π^{-1} be the interpretation from τ to σ given by reversing in a natural way the interpretation Π :

$$\psi_R := R(\bar{y}) \vee \bigvee_{\mathcal{F} \in \mathcal{F}} \bigvee_{R^{\mathcal{F}(\bar{y})} \text{ holds}} \exists \tilde{x} R_{\mathcal{F}}(\tilde{x}, \bar{y}) \wedge \epsilon(\tilde{x}, \bar{y})$$

In the above sentence \tilde{x} represent the elements of \mathcal{F} not present among \bar{y} and in $R_{\mathcal{F}}(\tilde{x}, \bar{y})$, the reader should understand that the variables \tilde{x}, \bar{y} are reordered in a suitable fashion. The sentence ϵ is a conjunction of equalities between variables among \tilde{x}, \bar{y} .

By construction, $\Pi^{-1} \circ \Pi$ is the identity over σ -structures.

Construction of the template of the CSP. We build the τ -structure \mathcal{T}_{Ω} as an extension of the σ -structure \mathcal{T} describing the colours of the forbidden patterns problem Ω . So on σ both structures agree and for every n -ary new symbol $R_{\mathcal{F}}$ and for every n -tuples of colours c_1, c_2, \dots, c_n we set $R_{\mathcal{F}}(c_1, c_2, \dots, c_n)$ to hold unless,

- it is explicitly forbidden by a pattern (\mathcal{F}, f) where $f(x_i) = c_i$; or,
- ★ the coloured structure (\mathcal{F}, f) is implicitly forbidden by (\mathcal{G}, g) in \mathcal{F}' where \mathcal{G} is a substructure of \mathcal{F} and g agrees with $f(x_i) = c_i$ where defined⁶.

Computational equivalence. By construction, the forbidden patterns problem Ω reduces to $\text{CSP}(-, \mathcal{T}_{\Omega})$ via the interpretation Π . The converse interpretation Π^{-1} is not a reduction in general. It is a reduction for the τ -structures that will “not change too much” under $\Pi \circ \Pi^{-1}$. More formally, let \mathcal{B} be the image of a τ -structure \mathcal{A} under $\Pi \circ \Pi^{-1}$. The monotonic nature of the interpretations means that \mathcal{A} is necessarily a substructure of \mathcal{B} and that we only need to show that if \mathcal{A} is a yes-instance then so is \mathcal{B} . The colouring certificate for \mathcal{A} will validate \mathcal{B} provided that if a new tuple involving $R_{\mathcal{G}}$ appeared in \mathcal{B} it is a consequence of a larger tuple $R_{\mathcal{F}}$ where \mathcal{F} and \mathcal{G} are patterns in \mathcal{F} and \mathcal{G} is a substructure of \mathcal{F} . This holds because of the condition ★ in the construction of \mathcal{T}_{Ω} .

In particular, we can guarantee that $\Pi \circ \Pi^{-1}$ will not change too much a τ -structure \mathcal{A} if it is of sufficiently high girth, say a girth higher than γ , the number of elements of the largest pattern in \mathcal{F} (this is because all patterns in \mathcal{F} are biconnected). This proves that Π^{-1} is a reduction for instances of girth greater or equal to γ .

Note that we may extend \mathcal{F} with any biconnected σ -structure without affecting the constructions or the result. This concludes the proof. □

⁶ This second case ★ allows to channel constraints from one symbol in τ to another as all information regarding the relationship between the forbidden patterns is lost in the new signature τ .

4 Recolouring Captures Containment

This section is a proof of our main result (Theorem 8).

Let \mathcal{F} be the set of biconnected structures involved as patterns in both Ω_1 and Ω_2 . Let γ be the size of the largest structure in \mathcal{F} . We use Theorem 11 for each problem, using \mathcal{F} as a parameter, and obtain a τ -structure \mathcal{T}_{Ω_1} for Ω_1 and a τ -structure \mathcal{T}_{Ω_2} for Ω_2 .

Lemma 13. *If Ω_1 is contained in Ω_2 then $\text{CSP}(\text{girth} > \gamma, \mathcal{T}_{\Omega_1})$ is contained in $\text{CSP}(\text{girth} > \gamma, \mathcal{T}_{\Omega_2})$.*

Proof. Let \mathcal{A} be a τ -structure of girth greater than γ such that there is a homomorphism from \mathcal{A} to \mathcal{T}_{Ω_1} . Since Π^{-1} is a reduction to Ω_1 , it follows that $\Pi^{-1}(\mathcal{A})$ is a yes-instance of Ω_1 . By inclusion of Ω_1 in Ω_2 it follows that $\Pi^{-1}(\mathcal{A})$ is also a yes-instance of Ω_2 . Since Π is a reduction from Ω_2 to $\text{CSP}(-, \mathcal{T}_{\Omega_2})$, the structure $\mathcal{B} := \Pi \circ \Pi^{-1}(\mathcal{A})$ is a yes-instance of $\text{CSP}(-, \mathcal{T}_{\Omega_2})$. Hence, there is a homomorphism from \mathcal{B} to \mathcal{T}_{Ω_2} . Since \mathcal{A} is a substructure of \mathcal{B} by monotonicity of the interpretations, it follows that there is a homomorphism from \mathcal{A} to \mathcal{T}_{Ω_2} . \square

Using Erdős Lemma we will derive the following.

Lemma 14. *The following are equivalent.*

- (i) $\text{CSP}(\text{girth} > \gamma, \mathcal{T}_{\Omega_1})$ is contained in $\text{CSP}(\text{girth} > \gamma, \mathcal{T}_{\Omega_2})$.
- (ii) $\text{CSP}(-, \mathcal{T}_{\Omega_1})$ is contained in $\text{CSP}(-, \mathcal{T}_{\Omega_2})$.
- (iii) There is a homomorphism from \mathcal{T}_{Ω_1} to \mathcal{T}_{Ω_2} .

Proof. The equivalence between (ii) and (iii) is easy and well known. The implication from (ii) to (i) holds trivially.

We prove that (i) implies (iii). Let \mathcal{D} be the structure obtained from Erdős Lemma from $\mathcal{B} := \mathcal{T}_{\Omega_1}$ with $s := |\mathcal{T}_{\Omega_2}|$ and $g := \gamma$. We know that there is a homomorphism from \mathcal{D} of girth greater than γ to \mathcal{T}_{Ω_1} . It follows from our assumption (i) that there is also a homomorphism from \mathcal{D} to \mathcal{T}_{Ω_2} . Appealing to Erdős Lemma again for $\mathcal{C} := \mathcal{T}_{\Omega_1}$, we finally have that there is a homomorphism from $\mathcal{B} = \mathcal{T}_{\Omega_1}$ to $\mathcal{C} = \mathcal{T}_{\Omega_2}$. \square

Lemma 15. *If r is a homomorphism from \mathcal{T}_{Ω_1} to \mathcal{T}_{Ω_2} then r is a recolouring from Ω_1 to Ω_2 .*

Proof. Recall that \mathcal{T}_1 (respectively \mathcal{T}_2) the structure used to colour the forbidden patterns of Ω_1 (respectively Ω_2) is by construction the σ -reduct of \mathcal{T}_{Ω_1} (respectively, \mathcal{T}_{Ω_2}). Hence, r is readily a homomorphism from \mathcal{T}_{Ω_1} to \mathcal{T}_{Ω_2} .

It remains to show that for any \mathcal{T}_2 -coloured pattern (\mathcal{F}_2, f_2) forbidden by Ω_2 , any of its inverse image under r —that is a \mathcal{T}_1 -coloured structure (\mathcal{F}_2, f_1) such that $f_2 = r \circ f_1$ —is not valid w.r.t. Ω_1 . Let (\mathcal{F}_2, f_2) and (\mathcal{F}_2, f_1) be as above. By construction of \mathcal{T}_{Ω_2} , the tuple $R_{\mathcal{F}_2}(f_2(\bar{x}))$ does not hold in \mathcal{T}_{Ω_2} . Since r is a homomorphism such that $f_2 = r \circ f_1$, the tuple $R_{\mathcal{F}_2}(f_1(\bar{x}))$ does not hold in \mathcal{T}_{Ω_1} . By construction of \mathcal{T}_{Ω_1} , this is because either a coloured pattern (\mathcal{G}_1, g_1) forbidden by Ω_1 with pattern \mathcal{F}_2 or a substructure of \mathcal{F}_2 disallowed this tuple. In any case, we have that (\mathcal{G}_1, g_1) , which is forbidden by Ω_1 occurs in (\mathcal{F}_2, f_1) . This shows that (\mathcal{F}_2, f_1) is not valid w.r.t. Ω_1 . \square

Our main result follows directly from the three previous lemmas.

Proof (of the main result). The definition of a recolouring implies containment as proved in Proposition 3. We now prove the converse. Suppose that Ω_1 is contained in Ω_2 . By Lemma 13, it follows that $\text{CSP}(\text{girth} > \gamma, \mathcal{T}_{\Omega_1})$ is contained in $\text{CSP}(\text{girth} > \gamma, \mathcal{T}_{\Omega_2})$. By Lemma 15, it follows that there is a homomorphism r from \mathcal{T}_{Ω_1} to \mathcal{T}_{Ω_2} . Finally, by Lemma 15 it follows that r is a recolouring from Ω_1 to Ω_2 . \square

We can strengthen our main result by relaxing some hypothesis as follows 7.

Corollary 16. *Let Ω_1 and Ω_2 be two forbidden patterns problems over the relational signature σ . If Ω_1 is given in a form that satisfies properties \mathbf{p}_1 to \mathbf{p}_5 then Ω_1 is contained in Ω_2 if, and only if, there is a recolouring from Ω_1 to Ω_2 .*

5 Closing Remarks

Feder and Vardi argued that MMSNP containment is decidable 11. However a precise complexity was not given. Every sentence of MMSNP captures a finite union of forbidden patterns problems 3 so this motivates us to reformulate the question in terms of forbidden patterns problems.

FPP-Containment:

- Input: forbidden patterns problems Ω_1 and Ω_2 given by $(\mathcal{T}_1, \mathcal{F}'_1)$ and $(\mathcal{T}_2, \mathcal{F}'_2)$.
- Question: is Ω_1 contained in Ω_2 ?

It is not difficult to see that the problem is at least NP-hard. Indeed, in the restricted case when the problems have no forbidden patterns, we have in fact the CSP-containment problem (also known as the uniform constraint satisfaction problem) which is NP-complete. In the restricted case when Ω_1 is given by a representation $(\mathcal{T}_1, \mathcal{F}'_1)$ which satisfies properties \mathbf{p}_1 to \mathbf{p}_5 , the question is equivalent to the following decision problem (see Corollary 16).

Recolouring:

- Input: forbidden patterns problems Ω_1 and Ω_2 given by $(\mathcal{T}_1, \mathcal{F}'_1)$ and $(\mathcal{T}_2, \mathcal{F}'_2)$.
- Question: is there a recolouring from Ω_1 to Ω_2 ?

The complexity of this problem is at most in Σ_3^p . This third level of the polynomial hierarchy is obtained directly from the definition of a recolouring. *Guess a homomorphism r , for every inverse image of every forbidden pattern, guess that it is non valid.* There are not many known complete problems in the third level of the polynomial hierarchy to choose from. There are however a myriad of problems in the second level. Using **Generalised Graph Colouring** 16 we can easily show that.

Proposition 17. *The restriction of **Recolouring** where Ω_2 has a single colour is Π_2^p -complete. Consequently, **Recolouring** is Π_2^p -hard.*

⁷ Note that this is the best we can do as we may not do without property \mathbf{p}_5 as example 7 shows.

In future work, we will try to pinpoint more accurately the complexity of **Recolouring**, which should be complete for Σ_3^P . Our hope is that a suitable generalisation of recoloring will enable us to derive that the complexity of **FPP-Containment** and **Recolouring** are the same. The long term aim is to classify the complexity of MMSNP containment. Though the translation to FPP is exponential, we hope that the insight gained in the combinatorial world of forbidden patterns problems can be used to solve the problem in the logical world of MMSNP.

References

1. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. *SIAM J. Comput.* 28, 57–104 (1999); An initial version of this paper appeared in *STOC* 1993
2. Bodirsky, M.: Constraint satisfaction problems with infinite templates. In: *Complexity of Constraints*, pp. 196–228 (2008)
3. Madelaine, F., Stewart, I.A.: Constraint satisfaction, logic and forbidden patterns. *SIAM Journal on Computing* 37(1), 132–163 (2007)
4. Ladner, R.E.: On the structure of polynomial time reducibility. *J. Assoc. Comput. Mach.* 22, 155–171 (1975)
5. Bulatov, A.A., Valeriote, M.: Recent results on the algebraic approach to the CSP. In: *Complexity of Constraints*, pp. 68–92 (2008)
6. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: Karp, R.M. (ed.) *Complexity of Computation*, pp. 43–73 (1974)
7. Kun, G.: Constraints, MMSNP and expander relational structures. arXiv:0706.1701v1 (June 2007)
8. Madelaine, F., Stewart, I.A.: Some problems not definable using structures homomorphisms. *Ars Combinatoria* LXVII (2003)
9. Madelaine, F.: Constraint satisfaction problems and related logic. PhD thesis, University of Leicester, Department of Maths and Computer Science (2003)
10. Bodirsky, M., Dalmau, V.: Datalog and constraint satisfaction with infinite templates. In: Durand, B., Thomas, W. (eds.) *STACS 2006*. LNCS, vol. 3884, pp. 646–659. Springer, Heidelberg (2006)
11. Madelaine, F.R.: Universal structures and the logic of forbidden patterns. *Logical Methods in Computer Science* 5(2) (2009)
12. Bodirsky, M., Chen, H., Feder, T.: On the complexity of MMSNP (November 2009)
13. Hell, P., Nešetřil, J.: *Graphs and homomorphisms*. Oxford University Press, Oxford (2004)
14. Cherlin, G., Shelah, S., Shi, N.: Universal graphs with forbidden subgraphs and algebraic closure. *Adv. Appl. Math.* 22(4), 454–491 (1999)
15. Foniok, J., Nešetřil, J., Tardif, C.: Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *Eur. J. Comb.* 29(4), 881–899 (2008)
16. Schaefer, M., Umans, C.: Completeness in the polynomial-hierarchy: part I. *SIGACT news* 33(3), 32–49 (2002); *SIGACT news complexity theory column* 37, Guest column, introduced by Lane A. Hemaspaandra

Improving the Floating Point Addition and Subtraction Constraints^{*}

Bruno Marre¹ and Claude Michel²

¹ CEA, LIST, Gif-sur-Yvette, F-91191 France

`Bruno.Marre@cea.fr`

² I3S (CNRS/UNSA)

930, Route des Colles - BP 145, 06903 Sophia Antipolis Cedex

`Claude.Michel@i3s.unice.fr`

Abstract. Solving constraints over the floating point numbers is a key issue in the process of software validation and verification. Techniques to solve such constraints on the basis of projection functions have been successfully developed. However, though correct, this approach can lead to slow convergence phenomena for very common constraints like addition and subtraction constraints. In this paper, we introduce new addition and subtraction constraints which, thanks to a new floating point subtraction property, directly compute optimal bounds for the domain of the variables at a low cost. Preliminary experiments have shown that these constraints can drastically speed up the filtering process.

1 Introduction

Constraint programming is one of the successful techniques applied to the validation and verification of software. It has been used to generate test cases [7,3] and functional test sequences [3], or to verify the conformity of a program with its specification [2]. For such applications, handling constraints over the floating point numbers is a critical issue as more and more critical software make use of floating point computations.

Constraints over the floating point numbers must not be mixed up with constraints over the reals. Both of them use intervals bounded by floating point numbers. However, constraints over the floating point numbers handle intervals of floating point numbers while constraints over the real numbers handle intervals of real numbers. Moreover, as underlined in [5,4], they do not have the same set of solutions¹. E.g., $16.0 + x > 16.0$, with $x < 0$, has no solution over the reals while it has numerous solutions over the floating point numbers. Solving constraints over the floating point numbers requires thus dedicated solvers.

Constraints over the floating point numbers have been solved using an adaptation of box-consistency [5] or the more efficient projection functions introduced

^{*} This work has been partially supported by the “CAVERN” ANR-07-SESUR-003 project.

¹ Even when restricted to floating point numbers.

in [4] and extended in [1]. All these projection functions compute a safe bounding of the domains of the variables. However, sometimes, the computed domains are rough over-approximations of the set of solutions. Moreover, these projections functions can led to slow convergence phenomena. Such behaviors occur for the two most common operations, the addition and the subtraction. To overcome this problem, we introduce here two new constraints, one for the addition and one for the subtraction, which, thanks to a new property of the subtraction, can drastically improve the set of computed solutions and speed up the filtering process.

Let us illustrate our motivations through a very simple example. Consider the addition constraint $z = x \oplus_{+\infty} y$, where $\oplus_{+\infty}$ is the floating point addition with a rounding mode set to $+\infty$ and z, x , and y are 80 bits floating point numbers as available on an Intel platform. Assume that, after a partial setting, $z = 2^-$, the predecessor of 2, and that \mathbf{x} , the domain of x , and \mathbf{y} , the domain of y , are restricted to $[-100.0, 100.0]$. Using the projection functions defined in [4], the solver attempts first to reduce the domain of z according to x and y :

$$\mathbf{z} = \mathbf{z} \cap [-200.0, 200.0] = [2^-, 2^-]$$

The domain of z being a degenerated interval, this projection can not reduce its domain. However, the reader may have noted that the bounds of the interval of possible values computed by the projection function, $[-200.0, 200.0]$, have actual support values in \mathbf{x} and \mathbf{y} . Inverse projections, i.e., trying to reduce the domain of x and y according to \mathbf{z} and, respectively, \mathbf{y} and \mathbf{x} , is more successful:

$$\begin{aligned} \mathbf{x} &= \mathbf{x} \cap [-98.0, 102.0^-] = [-98.0, 100.0] \\ \mathbf{y} &= \mathbf{y} \cap [-98.0, 100.0^-] = [-98.0, 100.0^-] \end{aligned}$$

At this step, two observations can be done. First, the computed intervals over-approximate the set of possible solutions as for e.g. $x = -98.0$ has no support in \mathbf{y} . Second, the upper bound of \mathbf{y} has been reduced from one floating point number when compared to the one of \mathbf{x} . This is where our slow convergence process takes its roots. Next iteration step of the filtering process will still reduce the domain of x and y :

$$\begin{aligned} \mathbf{x} &= \mathbf{x} \cap [-98.0^-, 100.0^-] = [-98.0^-, 100.0^-] \\ \mathbf{y} &= \mathbf{y} \cap [-98.0^-, (100.0^-)^-] = [-98.0^-, (100.0^-)^-] \end{aligned}$$

where each bound of x and y domains diminishes of exactly one floating point number. The filtering process will go on for hours², withdrawing one floating point number on each bound of the intervals at each iteration step, until it reaches the following fix point:

$$\mathbf{z} = [2.0^-, 2.0^-], \quad \mathbf{x} = [-2.0^-, 4.0^-], \quad \mathbf{y} = [-2.0^-, 4.0^-]$$

² Four hours of computation was not enough to read the fix point. Note that more than 10^{20} iterations are required to reach the fix point.

Note that the bounds of the resulting intervals, \mathbf{x} and \mathbf{y} , are solutions of the subtraction, i.e., $4.0^- \oplus_{+\infty} -2.0^- = 2.0^-$.

Projection functions as introduced in [4] have been built for unary functions. They rely on the correctly rounded property, an essential property to reason about floating point arithmetic which establishes a relation between an operation over the real numbers and its implementation over the floating point numbers. As a consequence, they take only into account the relationship between the floating point function and its corresponding real function. Though sufficient to handle unary functions, these projection functions tend to overestimate the domains of variables involved in binary functions. The new constraints we introduce here will not only overcome slow convergence issues but also offer a better bounding of the domains of the variables.

This paper is organised as follows: next section gives basic notations and definitions. Section 3 introduces a property of the subtraction and extends it to intervals. Finally, section 4 shows some experiments and is followed by a conclusion.

2 Notations and Definitions

The set of floating point numbers is a finite set of numbers usually used to simulate real numbers on a computer. In this paper, we restrict ourselves to the commonly available set of binary floating point numbers as described in the IEEE standard for floating point arithmetic revised in 2008. This section only recalls the basics of floating point arithmetic required to understand this paper.

A binary floating point number x is triple (s, e, m) where s is the sign bit of x , e , its exponent, and m , its significand (or mantissa). The value of x is given by:

$$x = (-1)^s * m * 2^e = (-1)^s * b_1.b_2 \dots b_p * 2^e$$

where the b_i are the bits which make up the significand and $b_2 \dots b_p$ denotes a sequence of $p-1$ bits. Note that p , the number of bits of the significand also gives the representation precision. In the sequel, \mathcal{F} will denote a set of binary floating point numbers and $\mathcal{F}_{p,q}$ will denote a set of binary floating point numbers where p is the number of bits of the significand and q the number of bits of the exponent. We will also use $\mathcal{F}_{p,+\infty}$, a set of floating point numbers whose significand has p bits and whose exponent has not limitation.

Operations on floating point numbers do not usually yield a floating point number. Floating point arithmetic resorts to rounding operations to get the most appropriate floating point representation according to a rounding direction. The standard requires that the four basic operations $\oplus, \ominus, \otimes, \oslash$ (which stand for, respectively, floating point addition, subtraction, multiplication, and division) are correctly rounded:

$$x \oplus y = \circ(x + y)$$

i.e., the result of the operation over the floating point numbers, an addition here, must be equal to the rounding, denoted by \circ , of the result of the same operation

over the real numbers. This property is a key property to reason about floating point numbers.

Domains of variables will be represented by intervals of floating point numbers:

$$\mathbf{x} = [\underline{x}, \overline{x}] = \{x \in \mathcal{F}, \underline{x} \leq x \leq \overline{x}\}$$

Note that, in this paper, unless otherwise explicitly stated, intervals will always be intervals of floating point numbers and not intervals of real numbers.

In the sequel, x^+ denotes the smallest floating point number superior to x while x^- denotes the biggest floating point number inferior to x .

3 Improving the Addition and Subtraction Constraints

3.1 Upper Bounds for the Subtraction

We introduce here a new property of the floating point subtraction which allows to compute upper bounds for its operands knowing its result. Intuitively, this property is based on the following observation: given a floating point number x , the smallest floating point number greater than 0 that can be obtained by subtraction to x is $x - x^-$. Note that the subtraction handled here is a true subtraction, i.e., that it actually subtracts x from y . These upper bounds for x and y , respectively, β and α , are given by :

Proposition 1. *Let $z \in \mathcal{F}_{p,+\infty}$, with $+\infty > z > 0$ and assume that*

$$z = 1.b_2 \dots b_i 0 \dots 0 * 2^{e_z} \text{ with } b_i = 1$$

$$\alpha = 1.1 \dots 1 * 2^{e_z + nb_z} \text{ with } nb_z = p - i$$

$$\beta = \alpha \oplus z$$

then, there is no $x \in \mathcal{F}_{p,+\infty}$, $x > \beta$ and no $y \in \mathcal{F}_{p,+\infty}$, $y > \alpha$ such that $x \ominus y = z$. Moreover, $\beta \ominus \alpha = \beta - \alpha = z$.

Proof. First, let us consider β , that is to say, $\alpha \oplus z$. Using a correctly rounded addition, $\alpha \oplus z = o(\alpha + z)$. Thus, the binary addition adds the two following numbers:

$$\alpha + z = 1.1 \dots 1 * 2^{e_z + nb_z} + 1.b_2 \dots b_{i-1} 10 \dots 0 * 2^{e_z}$$

The first step is to scale z to the same exponent than β :

$$\alpha + z = 1.1 \dots 1 * 2^{e_z + nb_z} + 0.0 \dots 01b_2 \dots b_{i-1} 1 * 2^{e_z + nb_z}$$

Splitting z on its lsb³ and the rest yields:

$$\begin{aligned} \alpha + z &= (1.1 \dots 1 * 2^{e_z + nb_z} + 0.0 \dots 01 * 2^{e_z + nb_z}) + 0.0 \dots 01b_2 \dots b_{i-1} * 2^{e_z + nb_z + 1} \\ &= 1.0 \dots 0 * 2^{e_z + nb_z + 1} + 0.0 \dots 01b_2 \dots b_{i-1} * 2^{e_z + nb_z + 1} \\ &= 1.0 \dots 01b_2 \dots b_{i-1} * 2^{e_z + nb_z + 1} \end{aligned}$$

³ lsb stands for least significant bit.

The result of this addition can be directly represented over $\mathcal{F}_{p,+\infty}$ (β requires $nb_z + i = (p - i) + i = p$ bits for its mantissa). Thus, $\beta = \alpha \oplus z = \alpha + z$ is computed exactly and so should be $\beta \ominus \alpha$.

Now, consider $\beta \ominus \alpha$. Using a correctly rounded subtraction, we can first do a binary subtraction:

$$\begin{aligned} \beta - \alpha &= 1.0 \dots 01b_2 \dots b_{i-1} * 2^{e_z+nb_z+1} - 1.1 \dots 1 * 2^{e_z+nb_z} \\ &= 1.0 \dots 01b_2 \dots b_{i-1} * 2^{e_z+nb_z+1} - 0.1 \dots 1|1 * 2^{e_z+nb_z+1} \end{aligned}$$

where $|$ separates the p representable bits from additional ones and the scaled α requires a guard bit $|1$ to be fully represented. Here, β is split on its msb⁴ yielding:

$$\begin{aligned} \beta - \alpha &= 0.0 \dots 01b_2 \dots b_{i-1} * 2^{e_z+nb_z+1} + (1.0 \dots 0 * 2^{e_z+nb_z+1} - 0.1 \dots 1|1 * 2^{e_z+nb_z+1}) \\ &= 0.0 \dots 01b_2 \dots b_{i-1} * 2^{e_z+nb_z+1} + 0.0 \dots 0|1 * 2^{e_z+nb_z+1} \\ &= 0.0 \dots 01b_2 \dots b_{i-1}|1 * 2^{e_z+nb_z+1} \\ &= 0.0 \dots 01b_2 \dots b_{i-1}1 * 2^{e_z+nb_z} \end{aligned}$$

which, after a normalization process directly yields z :

$$\beta - \alpha = 1.b_2 \dots b_{i-1}10 \dots 0 * 2^{e_z}$$

Like expected, $\beta \ominus \alpha = \beta - \alpha$ is computed exactly and yields z . Thus, the floating point numbers β and α provide a solution to $x \ominus y = z$.

The last issue is to prove that there is no $x' > \beta$ and no $y' > \alpha$ such that $x' \ominus y' = z$.

A first observation is that z is a solution of the subtraction: let $\dot{y} = z$, then $\dot{x} = \dot{y} \oplus z = o(z + z) = 2 * z = \dot{z}$, which belongs to $\mathcal{F}_{p,+\infty}$, and $\dot{x} \ominus \dot{y} = \dot{z} \ominus z = o(2 * z - z) = z$. Thus, if y' exists then $y' \geq z > 0$. Moreover, as $x' = y' \oplus z \leq y' \oplus y' = 2 * y'$ and, as $x' \ominus y' = z > 0$, we have

$$0 < z \leq y' \leq x' \leq 2 * y'$$

Notice that, thanks to these conditions, Sterbenz's property⁵ [6] applies and thus, if x' and y' exist, $x' \ominus y' = x' - y'$ and $y' \oplus z' = y' + z'$.

As $y' \leq x' \leq 2 * y'$, $e_{y'} \leq e_{x'} \leq e_{y'} + 1$. Therefore, scaling y' to x' requires at most one guard digit. α being the biggest floating point number whose exponent is equal to $e_z + nb_z$, if y' exists, $y' > \alpha$ and, $e_{y'} > e_z + nb_z$. As z mantissa requires $p - nb_z$ bits and as the scaling uses at most one guard digit, normalizing the result of the subtraction requires to add at most $nb_z + 1$ to its exponent. Thus, with $e_{y'} \geq e_z + nb_z + 1$, either $e_{x'} = e_{y'}$ and the normalization will remove nb_z to the exponent result (which is greater or equal to $e_z + nb_z + 1$), or $e_{x'} = e_{y'} + 1$ and

⁴ msb stands for most significant bit.

⁵ Sterbenz's property states that if the two floating point numbers x and y are such that $\frac{y}{2} \leq x \leq y$, then the subtraction $x \ominus y = x - y$, i.e., is exactly computed.

the normalization will remove $nb_z + 1$ to the exponent result (which is greater or equal to $e_z + nb_z + 2$). In both cases, the exponent result of the subtraction will be greater or equal to $e_z + 1$, and thus, is greater than z . □

For the sake of simplicity, proposition [11](#) relies on the idealized set of floating point numbers $\mathcal{F}_{p,+\infty}$ and, thus, does not take into account IEEE 754 exponent limitations. Here, two cases have to be handled: subnormal number and f_{max} , the biggest floating point number. As a matter of fact, proposition [11](#) also holds on subnormal numbers⁶ and thus, the only requirement is to ensure that neither β nor α are bigger than f_{max} .

3.2 Interval Extension of the Property

Proposition [11](#) sets upper bounds for x and y provided z is a strictly positive floating point number. In order to use it in a constraint solver, it now needs to be extended to intervals. Note that the property can easily be extended to negative values by symmetry. Therefore, this section handles only positive values for z .

When z belongs to an interval $\mathbf{z} = [\underline{z}, \overline{z}]$, with $\underline{z} > 0$, the subtraction upper bounds grow with respect to z exponent and the number of successive zero bits on the right of z mantissa, i.e., nb_z . Thus, upper bounds for the whole interval are given by ζ , the value of z which maximizes these two criteria.

Two cases can be distinguished: either $e_{\underline{z}} \neq e_{\overline{z}}$ or not. In the former case, $\zeta = 1.0 \dots 0 * 2^{e_{\overline{z}}}$ belongs to \mathbf{z} and is the floating point number with the highest exponent and the maximum number of binary zeros on its right. Therefore, this is the value that will have the biggest upper bounds for the subtraction. In the other case, i.e., if $e_{\underline{z}} = e_{\overline{z}}$, the floating point number with the highest upper bounds is the one with the most zeros on its right. Thus, either $\underline{z} = 1.0 \dots 0 * 2^{e_{\underline{z}}}$ and $\zeta = \underline{z}$, or $\zeta = 1.b_2 \dots b_i 10 \dots 0 * 2^{e_{\underline{z}}}$ where the $b_2 \dots b_i$ are identical leading bits of \underline{z} and \overline{z} . To sum up, we have:

$$\zeta = \begin{cases} 1.0 \dots 0 * 2^{e_{\overline{z}}} \text{ iff } e_{\underline{z}} \neq e_{\overline{z}} \\ \underline{z} \text{ iff } \underline{z} = 1.0 \dots 0 * 2^{e_{\underline{z}}} \\ 1.b_2 \dots b_i 10 \dots 0 * 2^{e_{\underline{z}}} \text{ with } \begin{cases} \underline{z} = b_1.b_2 \dots b_i 0 b_{i+2} \dots b_p * 2^{e_{\underline{z}}} \text{ and} \\ \overline{z} = b_1.b_2 \dots b_i 1 b'_{i+2} \dots b'_p * 2^{e_{\underline{z}}} \end{cases} & \text{otherwise} \end{cases}$$

Once ζ is known, α and β , i.e., upper bounds for a true subtraction of x and y , can be computed. As the bounds for a true addition are always lower than the one of the true subtraction, this result can easily be extended to the subtraction:

$$\text{if } z = x \ominus y \text{ then } \begin{cases} \mathbf{x} = \mathbf{x} \cap [-\alpha, \beta] \\ \mathbf{y} = \mathbf{y} \cap [-\beta, \alpha] \end{cases}$$

⁶ An essential observation here is that subnormal numbers can be handled without “normalizing” them i.e. $0.b_2 \dots b_p * 2^{-126}$ for simple floating point numbers. Then, the subtraction must preserve the same number of bits (i.e. $0.b_2 \dots b_i$).

```

const
  accuracy = 1e-8;
  max_move = 10.0;

node null_up_to_accuracy(V: real)
returns (null: Bool);
let
  null = -accuracy <= V and V <= accuracy;
tel;

node normal_move(In:real)
returns (OK:Bool);
var
  diff_previous: real;
let
  assert In >= 0.0 and In <= 1e40;

  diff_previous = 0.0 -> (In - pre(In));
  OK = if null_up_to_accuracy(diff_previous)
        then true
        else -max_move <= diff_previous and diff_previous <= max_move;
/*! reach OK and not(null_up_to_accuracy(diff_previous)) !*/

```

Fig. 1. A data acquisition procedure in Lustre

and to the addition:

$$\text{if } z = x \oplus y \text{ then } \begin{cases} \mathbf{x} = \mathbf{x} \cap [-\alpha, \beta] \\ \mathbf{y} = \mathbf{y} \cap [-\alpha, \beta] \end{cases}$$

Note that the computation of these projections are much more efficient than our previous projection functions, especially when a slow convergence occurs. However, classical projection functions will still improve the domain of x and y when the new projection function does not provide any reduction.

4 Preliminary Experiments

First, let us see what the projections introduced here can do on our initial example where $z = x \oplus_{+\infty} y$ with $z = 2^-$. z has no zero on its right. Thus, $\alpha = 1.1 \dots 1 * 2^{e_z} = 2^-$. As $\alpha = z$, computing β is easy: $\beta = \alpha \oplus z = 2 * \alpha = 1.1 \dots 1 * 2^{e_z+1} = 4^-$. As a consequence, $x \in [-2^-, 4^-]$ and $y \in [-2^-, 4^-]$ which is nothing but the fix point that has been reached after hours of computations.

Now, let us consider an experiment of test case generation from a data acquisition procedure written in LUSTRE, a declarative language often used to build synchronous reactive critical software. Figure 1 procedure `check` checks that data issued from a sensor evolve smoothly. The goal, defined in last algorithm line, is to generate test data that reach the else branch of the OK definition. Note that floating point variable type is double here. To reach this goal, GATeL [3] produces two different cases which negate “null_up_to_accuracy(diff_previous)”:

1. `diff_previous > 1e-8` : the domains of “In” and “PIn”, the variable associated to “pre(In)”, are both reduced to $[0.0, 1e40]$ while “diff_previous” is

⁷ In LUSTRE each variable denotes the data flow of its successive values at each cycle of reaction. Thus, expression “0.0 -> (In - pre(In))” yields 0.0 at the first computation cycle and the difference between the current value of “In” and its previous value at each other cycle.

reduced to $[1.0000000000000002e - 8, 10.0]$. Using classical projection functions, constraint “In - PIn = diff_previous” initiates a slow convergence process. Thanks to the subtraction property, “In” is immediately reduced to $[1.0000000000000002e - 8, 72057594037927936.0]$, while “Pin” is reduced to $[0.0, 72057594037927928.0]$ and “diff_previous” to $[1.0000000000000002e - 8, 10.0]$.

2. $\text{diff_previous} < -1e-8$: the domains of “In” and “PIn” are both reduced to $[0.0, 1e40]$ while “diff_previous” is reduced to $[-10.0, -1.0000000000000002e - 8]$. There is again a slow convergence linked to “In - PIn = diff_previous” constraint while the subtraction property allows an immediate reduction of “In” to $[0.0, 72057594037927928.0]$, “PIn” to $[1.0000000000000002e - 8, 72057594037927936.0]$ and “diff_previous” to $[-10, -1.0000000000000002e - 8]$.

These examples illustrate the benefit of the new projection functions which not only suppress the slow convergence phenomenon but also gives some solutions to the constraints. The new projection functions, which require far less than a millisecond of computation, drastically speed up the convergence process based on old projection functions which requires many hours of computation to reach its fix point.

5 Conclusion

In this paper, we have introduced new projection functions for the floating point addition and subtraction which, thanks to a property of the subtraction, drastically improve the solving of floating point constraints. The cost of these projections is negligible especially when compared to a slow convergence phenomenon. Preliminary experiments show the benefits of these projections and are really encouraging. Other constraints linked to multiplication and division are now studied to see if there is any possibility for such an improvement.

References

1. Botella, B., Gotlieb, A., Michel, C.: Symbolic execution of floating-point computations. *Softw. Test., Verif. Reliab.* 16(2), 97–121 (2006)
2. Collavizza, H., Rueher, M., Hentenryck, P.: CPBPV: a constraint-programming framework for bounded program verification. *Constraints* 15(2), 238–264 (2010)
3. Marre, B., Blanc, B.: Test selection strategies for lustre descriptions in gatel. *Electr. Notes Theor. Comput. Sci.* 111, 93–111 (2005)
4. Michel, C.: Exact projection functions for floating point number constraints. In: AMAI (2002)
5. Michel, C., Rueher, M., Lebbah, Y.: Solving constraints over floating-point numbers. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 524–538. Springer, Heidelberg (2001)
6. Sterbenz, P.: Floating point computations. Prentice-Hall, Englewood Cliffs (1974)
7. Williams, N., Marre, B., Mouy, P., Roger, M.: Pathcrawler: Automatic generation of path tests by combining static and dynamic analysis. In: Dal Cin, M., Kaâniche, M., Pataricza, A. (eds.) EDCC 2005. LNCS, vol. 3463, pp. 281–292. Springer, Heidelberg (2005)

The Lattice Structure of Sets of Surjective Hyper-Operations

Barnaby Martin*

School of Engineering and Computing Sciences, Durham University
Science Labs, South Road, Durham, DH1 3LE, UK
barnabymartin@gmail.com

Abstract. We study the lattice structure of sets (monoids) of surjective hyper-operations on an n -element domain. Through a Galois connection, these monoids form the algebraic counterparts to sets of relations closed under definability in positive first-order (fo) logic without equality. Specifically, for a countable set of relations (forming the finite-domain structure) \mathcal{B} , the set of relations definable over \mathcal{B} in positive fo logic without equality consists of exactly those relations that are invariant under the surjective hyper-endomorphisms (shes) of \mathcal{B} . The evaluation problem for this logic on a fixed finite structure is a close relative of the quantified constraint satisfaction problem (QCSP).

We study in particular an inverse operation that specifies an automorphism of our lattice. We use our results to give a dichotomy theorem for the evaluation problem of positive fo logic without equality on structures that are *she-complementative*, i.e. structures \mathcal{B} whose set of shes is closed under inverse. These problems turn out either to be in \mathbf{L} or to be \mathbf{Pspace} -complete.

We go on to apply our results to certain digraphs. We prove that the evaluation of positive fo without equality on a semicomplete digraph is always \mathbf{Pspace} -complete. We go on to prove that this problem is \mathbf{NP} -hard for any graph of diameter at least 3. Finally, we prove a tetrachotomy for antireflexive and reflexive graphs, modulo a known conjecture as to the complexity of the QCSP on connected non-bipartite graphs. Specifically, these problems are either in \mathbf{L} , \mathbf{NP} -complete, $\mathbf{co-NP}$ -complete or \mathbf{Pspace} -complete.

1 Introduction

We continue the study of the evaluation problem for positive equality-free first-order (fo) logic, on a fixed finite structure \mathcal{B} , denoted $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}), started in [5]. This problem is a close relative of the *constraint satisfaction problem*, CSP(\mathcal{B}), and an even closer relative of the *quantified CSP*, QCSP(\mathcal{B}). In fact, it is noted in [5] that among a wide family of problems, the only interesting case, other than the CSP and QCSP, is the one addressed in this paper. The bulk of the theoretical research into CSPs concerns the so-called *dichotomy conjecture*:

* The author is supported by EPSRC grant EP/G020604/1.

that the complexity of the problem of evaluating a primitive positive sentence on a fixed finite \mathcal{B} , $\text{CSP}(\mathcal{B})$, is either in P or is NP -complete. This was solved for structures with two-element domains in [11] and improved to encompass structures with three-element domains in [3]. The most successful approach to date, and the method used in [3], has been the so-called algebraic method, in which the problem of classification reverts to classes of functions under which the relevant relational systems are invariant. A similar algebraic approach has been successful in the study of the evaluation problem for positive Horn sentences, $\text{QCSP}(\mathcal{B})$, and, while no formal trichotomy has there been conjectured, the only known attainable complexities are P , NP -complete and Pspace -complete (see trichotomies in [28]).

In [5], the complexity of $\{\exists, \forall, \wedge, \vee\}$ - $\text{FO}(\mathcal{B})$ is studied through an analogous algebraic method to that used for $\text{CSP}(\mathcal{B})$. The paper culminates in a full classification – a tetrachotomy – as \mathcal{B} ranges over structures with three-element domains. Specifically, the problems $\{\exists, \forall, \wedge, \vee\}$ - $\text{FO}(\mathcal{B})$ are either Pspace -complete, NP -complete, co-NP -complete or in L . In a recent paper [9] this tetrachotomy is extended to four-element domains.

These results come from studying sets of surjective hyper-operations (shops) under which the relational \mathcal{B} may be invariant. These sets, always containing the identity and closed under composition and sub-shops, are known as *down shop-monoids* (DSMs). In this paper we study the structure of the lattice of DSMs on a k -element domain. We give a full Galois connection proving an isomorphism between the lattice of DSMs (over a k -element domain) and the lattice of k -element structures closed under definability in positive fo without equality (in previous papers, only the relational side of this connection appeared). We study in particular the automorphism born of the inverse operation on shops. DSMs that are closed under inverse have a fundamentally group-like structure – what we call *blurred permutation subgroups* (BPSs). Using this characterisation, we prove a dichotomy for our evaluation problem on structures that we term *she-complementative*, i.e whose set of surjective hyper-endomorphisms (shes) is closed under inverse. Specifically, these problems are either in L or are Pspace -complete. We conclude with some natural examples of she-complementative classes, namely tournaments and structures that for each of their relations R contain also the complement \overline{R} .

We go on to use our results in the study of $\{\exists, \forall, \wedge, \vee\}$ - $\text{FO}(\mathcal{H})$ where \mathcal{H} is a semicomplete digraph – extending the result for tournaments – or a certain type of graph. In the first case, where \mathcal{H} is a semicomplete digraph, $\{\exists, \forall, \wedge, \vee\}$ - $\text{FO}(\mathcal{H})$ is proved to be Pspace -complete. In the case where \mathcal{H} is a graph of diameter at least 3 we prove that $\{\exists, \forall, \wedge, \vee\}$ - $\text{FO}(\mathcal{H})$ is NP -hard (this result is optimal in the sense that there is a graph of diameter 2 such that the problem is in co-NP). Finally, when \mathcal{H} is an antireflexive graph or a reflexive graph, we rely on a conjecture from QCSP in [8] to derive our classification. Modulo that conjecture we derive that the problem $\{\exists, \forall, \wedge, \vee\}$ - $\text{FO}(\mathcal{H})$ is either in L (if \mathcal{H} is trivial), is NP -complete (if \mathcal{H} is non-trivial with an isolated vertex), is co-NP -complete (if \mathcal{H} is non-trivial with a dominating vertex) and is Pspace -complete, otherwise.

The paper is organised as follows. In Section 2 we give the necessary preliminaries and introduce the Galois connection. In Section 3, we discuss the structure of our lattices, with particular emphasis on an automorphism born of an inverse operation. We go on to prove the main characterisation theorem and use it to derive our complexity dichotomy for she-complementative structures. In Section 4, we go on to prove the further complexity results for digraphs and in Section 5, we make some final remarks.

2 Preliminaries

Throughout, let \mathcal{B} be a finite structure, with domain B , over an at most countable relational signature σ . Let $\{\exists, \forall, \wedge, \vee\}$ -FO be the positive fragment of first-order (fo) logic without equality. An *extensional* relation is one that appears in the signature σ . We will usually denote extensional relations of \mathcal{B} by R and other relations by S (or by some formula that defines them). In $\{\exists, \forall, \wedge, \vee\}$ -FO the atomic formulae are exactly substitution instances of extensional relations. The problem $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) has:

- Input: a sentence $\varphi \in \{\exists, \forall, \wedge, \vee\}$ -FO.
- Question: does $\mathcal{B} \models \varphi$?

QCSP(\mathcal{B}) is the restriction of this problem to formulae involving no disjunction, what in our notation would be $\{\exists, \forall, \wedge\}$ -FO. When \mathcal{B} is of size one, the evaluation of any FO sentence may be accomplished in L (essentially, the quantifiers are irrelevant and the problem amounts to the *boolean sentence value problem*, see [4]). In this case, it follows that $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) is in L. Furthermore, by inward evaluation of the quantifiers, $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) is readily seen to always be in Pspace.

For a structure \mathcal{B} define the complement structure $\overline{\mathcal{B}}$ to be over the same domain B with relations which are the set-theoretic complements of those of \mathcal{B} . That is, for each r -ary R , $R^{\overline{\mathcal{B}}} = B^r \setminus R^{\mathcal{B}}$. Similarly, for a relation $R \subseteq B^r$, let \overline{R} denote $B^r \setminus R$.

Consider the finite set $D = [n] := \{1, \dots, n\}$ and its power set $\mathfrak{P}(D)$. A *hyper-operation* on D is a function $f : D \rightarrow \mathfrak{P}(D) \setminus \{\emptyset\}$ (that the image may not be the empty set corresponds to the hyper-operation being *total*, in the parlance of [1]). If the hyper-operation f has the additional property that

- for all $y \in D$, there exists $x \in D$ such that $y \in f(x)$,

then we designate (somewhat abusing terminology) f *surjective*. A surjective hyper-operation (shop) in which each element is mapped to a singleton set is identified with a *permutation* (bijection). A *surjective hyper-endomorphism* (she) of a set of relations (forming the finite-domain structure) \mathcal{B} over D is a shop f on D that satisfies, for all relations R of \mathcal{B} ,

- if $(x_1, \dots, x_i) \in R$ then, for all $y_1 \in f(x_1), \dots, y_i \in f(x_i)$, $(y_1, \dots, y_i) \in R$.

More generally, for $r_1, \dots, r_k \in D$, we say f is a she from $(\mathcal{B}; r_1, \dots, r_k)$ to $(\mathcal{B}; r'_1, \dots, r'_k)$ if f is a she of \mathcal{B} and $r'_1 \in f(r_1), \dots, r'_k \in f(r_k)$. A she may be identified with a *surjective endomorphism* if each element is mapped to a singleton set. On finite structures surjective endomorphisms are necessarily automorphisms.

2.1 Galois Connections

Relational side. For a set F of shops on the finite domain B , let $\text{Inv}(F)$ be the set of relations on B of which each $f \in F$ is a she (when these relations are viewed as a structure over B). We say that $S \in \text{Inv}(F)$ is invariant or is *preserved* by (the shops in) F . Let $\text{shE}(\mathcal{B})$ be the set of shes of \mathcal{B} . Let $\text{Aut}(\mathcal{B})$ be the set of automorphisms of \mathcal{B} .

Let $\langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}}$ and $\langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee, =\}\text{-FO}}$ be the sets of relations that may be defined on \mathcal{B} in $\{\exists, \forall, \wedge, \vee\}\text{-FO}$ and $\{\exists, \forall, \wedge, \vee, =\}\text{-FO}$, respectively.

Lemma 1 ([5]). *Let $\mathbf{r} := (r_1, \dots, r_k)$ be a k -tuple of elements of the finite-signature \mathcal{B} . There exists:*

- (i). *a formula $\theta_{\mathbf{r}}(u_1, \dots, u_k) \in \{\exists, \forall, \wedge, \vee, =\}\text{-FO}$ s.t. $(\mathcal{B}, r'_1, \dots, r'_k) \models \theta_{\mathbf{r}}(u_1, \dots, u_k)$ iff there is an automorphism from $(\mathcal{B}, r_1, \dots, r_k)$ to $(\mathcal{B}, r'_1, \dots, r'_k)$.*
- (ii). *a formula $\theta_{\mathbf{r}}(u_1, \dots, u_k) \in \{\exists, \forall, \wedge, \vee\}\text{-FO}$ s.t. $(\mathcal{B}, r'_1, \dots, r'_k) \models \theta_{\mathbf{r}}(u_1, \dots, u_k)$ iff there is a she from $(\mathcal{B}, r_1, \dots, r_k)$ to $(\mathcal{B}, r'_1, \dots, r'_k)$.*

Proof. For Part (i), let $b_1, \dots, b_{|B|}$ an enumeration of the elements of \mathcal{B} and $\Phi_{\mathcal{B}}(v_1, \dots, v_{|B|})$ be the associated conjunction of positive facts. Set $\theta_{\mathbf{r}}(u_1, \dots, u_k) :=$

$$\exists v_1, \dots, v_{|B|} \Phi_{\mathcal{B}}(v_1, \dots, v_{|B|}) \wedge \forall v (v = v_1 \vee \dots \vee v = v_{|B|}) \wedge u_1 = v_{\lambda_1} \wedge \dots \wedge u_k = v_{\lambda_k},$$

where $r_1 = b_{\lambda_1}, \dots, r_k = b_{\lambda_k}$. The forward direction follows since \mathcal{B} is finite, so any surjective endomorphism is necessarily an automorphism. The backward direction follows since all fo formulae are preserved by automorphism.

[Part (ii).] This will require greater dexterity. Let $\mathbf{r} \in B^k$, $\mathbf{s} := (b_1, \dots, b_{|B|})$ be an enumeration of B and $\mathbf{t} \in B^{|B|}$. Recall that $\Phi_{\mathcal{B}(\mathbf{r}, \mathbf{s})}(u_1, \dots, u_k, v_1, \dots, v_{|B|})$ is a conjunction of the positive facts of (\mathbf{r}, \mathbf{s}) , where the variables (\mathbf{u}, \mathbf{v}) correspond to the elements (\mathbf{r}, \mathbf{s}) . In a similar manner, $\Phi_{\mathcal{B}(\mathbf{r}, \mathbf{s}, \mathbf{t})}(u_1, \dots, u_k, v_1, \dots, v_{|B|}, w_1, \dots, w_{|B|})$ is the conjunction of the positive facts of $(\mathbf{r}, \mathbf{s}, \mathbf{t})$, where the variables $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ correspond to the elements $(\mathbf{r}, \mathbf{s}, \mathbf{t})$. Set $\theta_{\mathbf{r}}(u_1, \dots, u_k) :=$

$$\exists v_1, \dots, v_{|B|} \Phi_{\mathcal{B}(\mathbf{r}, \mathbf{s})}(u_1, \dots, u_k, v_1, \dots, v_{|B|}) \wedge \forall w_1 \dots w_{|B|} \bigvee_{\mathbf{t} \in B^{|B|}} \Phi_{\mathcal{B}(\mathbf{r}, \mathbf{s}, \mathbf{t})}(u_1, \dots, u_k, v_1, \dots, v_{|B|}, w_1, \dots, w_{|B|}).$$

See [5] for a full proof of correctness.

The following is the main theorem of [5].

Theorem 1 ([5]). *For a finite-signature structure \mathcal{B} we have*

- (i). $\langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee, =\}\text{-FO}} = \text{Inv}(\text{Aut}(\mathcal{B}))$ and
- (ii). $\langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}} = \text{Inv}(\text{shE}(\mathcal{B}))$.

Proof. Part (i) is well-known and may be proved in a similar, albeit simpler, manner to Part (ii), which we now prove.

$[\varphi(\mathbf{v}) \in \langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}} \Rightarrow \varphi(\mathbf{v}) \in \text{Inv}(\text{shE}(\mathcal{B}))]$ This is proved by induction on the complexity of $\varphi(\mathbf{v})$ (see [5] for a full exposition).

$[S \in \text{Inv}(\text{shE}(\mathcal{B})) \Rightarrow S \in \langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}}]$ Consider the k -ary relation $S \in \text{Inv}(\text{shE}(\mathcal{B}))$. Let $\mathbf{r}_1, \dots, \mathbf{r}_m$ be the tuples of S . Set

$$\theta_S(u_1, \dots, u_k) := \theta_{\mathbf{r}_1}(u_1, \dots, u_k) \vee \dots \vee \theta_{\mathbf{r}_m}(u_1, \dots, u_k).$$

Manifestly, $\theta_S(u_1, \dots, u_k) \in \{\exists, \forall, \wedge, \vee\}\text{-FO}$. For $\mathbf{r}_i := (r_{i1}, \dots, r_{ik})$, note that $(\mathcal{B}, r_{i1}, \dots, r_{ik}) \models \theta_{\mathbf{r}_i}(u_1, \dots, u_k)$ (the ‘identity’ shop will be formally introduced in the next section). That $\theta_S(u_1, \dots, u_k) = S$ now follows from Part (ii) of Lemma [1], since $S \in \text{Inv}(\text{shE}(\mathcal{B}))$.

Theorem 2. *For a countable-signature structure \mathcal{B} we have*

- (i). $\langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee, =\}\text{-FO}} = \text{Inv}(\text{Aut}(\mathcal{B}))$ and
- (ii). $\langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}} = \text{Inv}(\text{shE}(\mathcal{B}))$.

Proof. Again, Part (i) is well-known and may be proved in a similar, but simpler, manner to Part (ii), which we now prove. The direction $[\varphi(\mathbf{v}) \in \langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}} \Rightarrow \varphi(\mathbf{v}) \in \text{Inv}(\text{shE}(\mathcal{B}))]$ is proved as before.

For $[S \in \text{Inv}(\text{shE}(\mathcal{B})) \Rightarrow S \in \langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}}]$, we proceed similarly to before, but using finiteness of the domain B , which will rescue us from the pitfalls of an infinite signature. Consider the finite disjunction we previously built:

$$\theta_S(u_1, \dots, u_k) := \theta_{\mathbf{r}_1}(u_1, \dots, u_k) \vee \dots \vee \theta_{\mathbf{r}_m}(u_1, \dots, u_k).$$

Let R_1, R_2, \dots be an enumeration of the extensional relations of \mathcal{B} . Let \mathcal{B}_i be the reduct of \mathcal{B} to the signature $\langle R_1, \dots, R_i \rangle$. For $j \in [m]$ let $\theta_{\mathbf{r}_j}^i(u_1, \dots, u_k)$ be built as in Lemma [1], but on the reduct \mathcal{B}_i . The relations $\theta_{\mathbf{r}_j}^1(u_1, \dots, u_k)$, $\theta_{\mathbf{r}_j}^2(u_1, \dots, u_k), \dots$ are monotone decreasing on B^k – the shes must preserve an increasing number of extensional relations – and therefore reach a limit l_j s.t. $\theta_{\mathbf{r}_j}^{l_j}(u_1, \dots, u_k) = \theta_{\mathbf{r}_j}(u_1, \dots, u_k)$. Let $l := \max\{l_1, \dots, l_m\}$ and build $\theta_S(u_1, \dots, u_k)$ over the finite-signature reduct \mathcal{B}_l . The result follows.

In the following, \leq_L indicates the existence of a logspace many-to-one reduction.

Theorem 3. *Let \mathcal{B} and \mathcal{B}' be structures over the same domain B s.t. \mathcal{B}' is finite-signature.*

- (i). *If $\text{Aut}(\mathcal{B}) \subseteq \text{Aut}(\mathcal{B}')$ then $\{\exists, \forall, \wedge, \vee, =\}\text{-FO}(\mathcal{B}') \leq_L \{\exists, \forall, \wedge, \vee, =\}\text{-FO}(\mathcal{B})$.*
- (ii). *If $\text{shE}(\mathcal{B}) \subseteq \text{shE}(\mathcal{B}')$ then $\{\exists, \forall, \wedge, \vee\}\text{-FO}(\mathcal{B}') \leq_L \{\exists, \forall, \wedge, \vee\}\text{-FO}(\mathcal{B})$.*

Proof. Again, Part (i) is well-known and the proof is similar to that of Part (ii), which we give. If $\text{shE}(\mathcal{B}) \subseteq \text{shE}(\mathcal{B}')$, then $\text{Inv}(\text{shE}(\mathcal{B}')) \subseteq \text{Inv}(\text{shE}(\mathcal{B}))$. From Theorem 11, it follows that $\langle \mathcal{B}' \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}} \subseteq \langle \mathcal{B} \rangle_{\{\exists, \forall, \wedge, \vee\}\text{-FO}}$. Recalling that \mathcal{B}' contains only a finite number of extensional relations, we may therefore effect a logspace reduction from $\{\exists, \forall, \wedge, \vee\}\text{-FO}(\mathcal{B}')$ to $\{\exists, \forall, \wedge, \vee\}\text{-FO}(\mathcal{B})$ by straightforward substitution of predicates.

Down-shop-monoids and the functional side. Consider the finite domain D . The *identity* shop id_D is defined by $x \mapsto \{x\}$. Given shops f and g , define the *composition* $g \circ f$ by $x \mapsto \{z : \exists y z \in g(y) \wedge y \in f(x)\}$. Finally, a shop f is a *sub-shop* of g – denoted $f \subseteq g$ – if $f(x) \subseteq g(x)$, for all x . A set of surjective shops on a finite set B is a *down-shop-monoid* (DSM), if it contains id_D , and is closed under composition and sub-shops¹ (of course, not all sub-hyper-operations of a shop are surjective – we are only concerned with those that are). id_B is a she of all structures with domain B , and, if f and g are shes of \mathcal{B} , then so is $g \circ f$. Further, if g is a she of \mathcal{B} , then so is f for all (surjective) $f \subseteq g$. It follows that $\text{shE}(\mathcal{B})$ is always a DSM. If F is a set of permutations, then we write $\langle F \rangle_G$ to denote the group generated by F . If F is a set of shops on B , then let $\langle F \rangle_{DSM}$ denote the minimal DSM containing the operations of F . If F is the singleton $\{f\}$, then, by abuse of notation, we write $\langle f \rangle$ instead of $\langle \{f\} \rangle$. We will mark-up, e.g., the shop $1 \mapsto \{1, 2\}$, $2 \mapsto \{2\}$, $3 \mapsto \{1, 3\}$ as $\frac{112}{212}$ /₃₁₃.

For a shop f , define its inverse f^{-1} by $x \mapsto \{y : x \in f(y)\}$. Note that f^{-1} is also a shop and $(f^{-1})^{-1} = f$, though $f \circ f^{-1} = id_B$ only if f is a permutation. For a set of shops F , let $F^{-1} := \{f^{-1} : f \in F\}$.

A *permutation subgroup* on a finite set B is a set of permutations of B closed under composition. It may easily be verified that such a set contains the identity and is closed under inverse. A permutation subgroup may be identified with a particular type of DSM in which all shops have only singleton sets in their range.

Theorem 4. *Let F be a set of permutations (Part (i)) or shops (Part (ii)) on the finite domain D . Then*

- (i) $\langle F \rangle_G = \text{Aut}(\text{Inv}(F))$, and
- (ii) $\langle F \rangle_{DSM} = \text{shE}(\text{Inv}(F))$.

Proof. Part (i) is well-known but we give a proof for illustrative purposes.

[$\langle F \rangle_G \subseteq \text{Aut}(\text{Inv}(F))$.] By induction. One may easily see that if $f, g \in \text{Aut}(\text{Inv}(F))$ then $f \circ g \in \text{Aut}(\text{Inv}(F))$. Further, if $f \in \text{Aut}(\text{Inv}(F))$ then $f^{-1} \in \text{Aut}(\text{Inv}(F))$ as the set of automorphisms is closed under inverse.

[$\text{Aut}(\text{Inv}(F)) \subseteq \langle F \rangle_G$.] Let $|D| = n$. One may easily see that $\text{Inv}(F) = \text{Inv}(\langle F \rangle_G)$ (for the forward containment, note that inverse follows from the fact that F is a set of bijections on a finite set). Let R be the n -ary relation that lists the permutations in $\langle F \rangle_G$ (e.g., the identity appears as $(1, 2, \dots, n)$); R is preserved by $\langle F \rangle_G$. We will prove $\text{Aut}(\text{Inv}(\langle F \rangle_G)) \subseteq \langle F \rangle_G$ by contraposition. If g

¹ Closure under sub-shops is termed *down closure* in [11], hence the D in DSM.

is a permutation not in $\langle F \rangle_G$, then $g \notin R$ and g does not preserve R as it maps the identity to g . Therefore $g \notin \text{Aut}(\text{Inv}(\langle F \rangle_G))$ and the result follows.

[Part (ii).]

$[\langle F \rangle_{DSM} \subseteq \text{shE}(\text{Inv}(F))]$ By induction. One may easily see that if $f, g \in \text{shE}(\text{Inv}(F))$ then $f \circ g \in \text{shE}(\text{Inv}(F))$. Similarly for sub-shops and the identity.

$[\text{shE}(\text{Inv}(F)) \subseteq \langle F \rangle_{DSM}]$ Let $|D| = n$. One may easily see that $\text{Inv}(F) = \text{Inv}(\langle F \rangle_{DSM})$. Let R be the n^2 -ary relation that lists the shes of $\langle F \rangle_{DSM}$ in the following manner. Consider the n^2 positions enumerated in n -ary, i.e. by (i, j) s.t. $i, j \in [n]$. Each she f gives rise to many tuples in which the positions $(i, 1), \dots, (i, n)$ are occupied in all possible ways by the elements of $f(i)$. Thus, $f_0 := \frac{11^2}{2^2 3^3}$ generates the following eight tuples

- (1, 1, 1, 2, 2, 2, 3, 3, 3)
- (1, 1, 2, 2, 2, 2, 3, 3, 3)
- (1, 2, 1, 2, 2, 2, 3, 3, 3)
- (1, 2, 2, 2, 2, 2, 3, 3, 3)
- (2, 1, 1, 2, 2, 2, 3, 3, 3)
- (2, 1, 2, 2, 2, 2, 3, 3, 3)
- (2, 2, 1, 2, 2, 2, 3, 3, 3)
- (2, 2, 2, 2, 2, 2, 3, 3, 3)

Let $p_{i,j} \in [n]$ be the element at position (i, j) . We describe as a *full coding* of f any such tuple s.t., for all i , $\{p_{i,1}, \dots, p_{i,|D|}\} = f(i)$. In our example, all tuples except the first and last are full codings of f_0 . Note that R is preserved by $\langle F \rangle_{DSM}$. We will prove $\text{shE}(\text{Inv}(\langle F \rangle_G)) \subseteq \langle F \rangle_{DSM}$ by contraposition. If g is a shop not in $\langle F \rangle_{DSM}$, then g does not appear fully coded in R and g does not preserve R as it maps the identity to all tuples that are full codings of g . Therefore $g \notin \text{shE}(\text{Inv}(\langle F \rangle_{DSM}))$ and the result follows.

2.2 Lattice Isomorphism

Consider sets of relations Γ on the normalised domain $D = [n]$, closed under $\{\exists, \forall, \wedge, \vee\}$ -FO-definability (such sets may be seen as countable signature structures \mathcal{D}). Let \mathcal{R}_n be the lattice of such sets ordered by inclusion. Let the lattice \mathcal{F}_n be of DSMs on the set $[n]$, again ordered by inclusion.

Corollary 1. *The lattices \mathcal{R}_n and \mathcal{F}_n are isomorphic and the operators Inv and shE induce isomorphisms between them.*

Proof. From the second parts of Theorems 2 and 4

The permutation subgroups form a lattice under inclusion whose minimal element contains just the identity and whose maximal element is the symmetric group $S_{|B|}$. As per Theorem 3, this lattice classifies the complexities of $\{\exists, \forall, \wedge, \vee, =\}$ -FO(\mathcal{B}) (again there is an isomorphism between this lattice and sets of relations closed under positive fo-definability). In the lattice of DSMs,

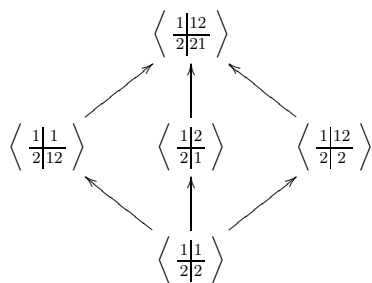


Fig. 1. The lattice \mathcal{F}_2

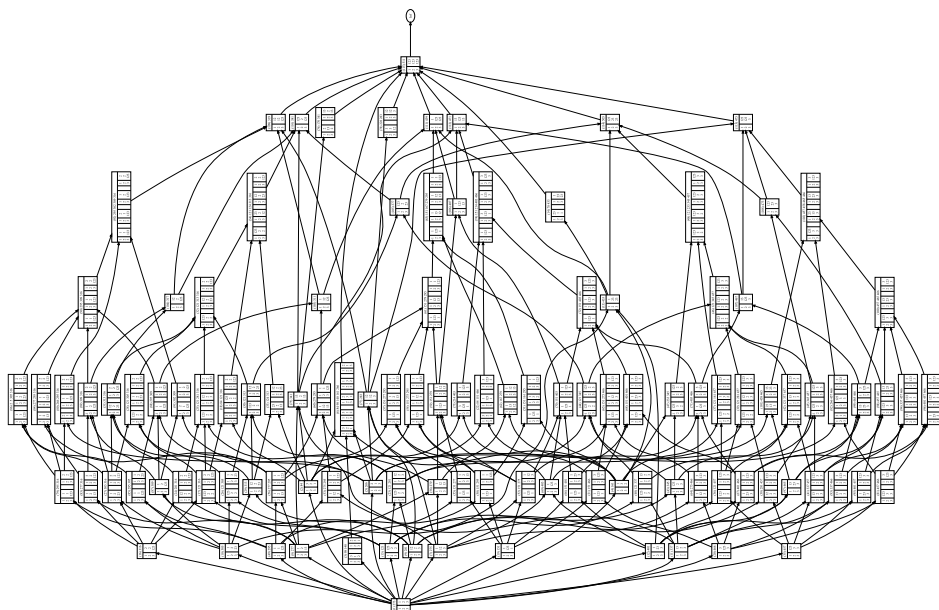


Fig. 2. The lattice \mathcal{F}_3 . At the bottom is the DSM containing only the identity, at the top is the DSM containing all shops. The author is grateful to his brother for calculating and drawing \mathcal{F}_3 . The circular node at the top of the diagram is superfluous to the lattice, being a figment of the imagination of the graph drawing package.

\mathcal{F}_n , the minimal element still contains just id_B , but the maximal element contains all shops. However, the lattice of permutation subgroups always appears as a sub-lattice within the lattice of DSMs. In the case of \mathcal{F}_2 , Figure 1, we have 5 DSMs, two of which are the subgroups of S_2 . In the case of \mathcal{F}_3 , Figure 2, we have 115 DSMs, only six of which are the subgroups of S_3 – so the lattice complexity jumps very quickly.

3 The Structure of \mathcal{F}_n

3.1 Blurred Permutation Subgroups and Diagonals

Many of the DSMs of \mathcal{L}_n are reminiscent of subgroups of the symmetric group S_m for some $m \leq n$. We say that a shop f on the domain $[n]$ is a *blurred permutation* if it may be built from (the shop associated with) the permutation g on the domain $[m]$ ($m \leq n$) in the following manner.

(*) Let P_1, \dots, P_m be a partition of $[n]$ s.t. $i \in P_i$ for $i \in [m]$ and each P_i may be listed $d_{i,1}, \dots, d_{i,l_i}$. Set $f(d_{i,1}) = \dots = f(d_{i,l_i}) = P_{g(i)}$.

We say that a DSM N over domain $[n]$ is a *blurred permutation subgroup* (BPS) if one may build it from (the DSM associated with) a subgroup M of S_m , $m \leq n$ by replacing each permutation $g \in M$ by the blurred permutation f created as in (*), and then taking the closure under sub-shops. We do not feel it necessary to elaborate on this construction save for the example that the group $M := \langle \frac{1|2}{2|1} \rangle$

- becomes the BPS $N := \langle \frac{1|234}{2|1} \rangle$ when $P_1 := \{1\}$ and $P_2 := \{2, 3, 4\}$, and
- becomes the BPS $N := \langle \frac{1|24}{2|13} \rangle$ when $P_1 := \{1, 3\}$ and $P_2 := \{2, 4\}$,

and the permutation $\frac{1|1}{2|3}$ becomes the blurred permutation $\frac{1|1}{2|34}$ when $P_1 := \{1\}$, $P_2 := \{2\}$ and $P_3 := \{3, 4\}$. From a blurred permutation (respectively, BPS) one may read the partitions P_1, \dots, P_m , for some suitably chosen permutation (respectively, group) on domain $[m]$. A *blurred symmetric group* is a BPS built in the manner described from a symmetric group.

With an arbitrary shop f on D , we may associate the digraph \mathcal{G}_f on D in which there is an edge (x, y) if $f(x) \ni y$. The condition of totality ensures \mathcal{G}_f has no sinks and the condition of surjectivity ensures \mathcal{G}_f has no sources. f contains the identity as a sub-shop iff \mathcal{G}_f is reflexive. There is an edge from a to some y in \mathcal{G}_g and an edge from y to b in \mathcal{G}_f iff there is an edge from a to b in $\mathcal{G}_{f \circ g}$. In this fashion, it is easy to verify that there is a directed path of length n from a to b in \mathcal{G}_f iff $b \in f^n(a)$.

Call a shop f *diagonal* if, for all a and b , we have $a \in f(b)$ iff $b \in f(a)$. Examples of diagonal shops are id_D and $\frac{1|12}{2|13}$. It not hard to see that f is diagonal iff $f = f^{-1}$ iff \mathcal{G}_f is symmetric (undirected). Diagonal shops are blurred permutations built in the manner (*) from an identity shop.

Lemma 2. *For all shops f , $f \circ f^{-1} = f^{-1} \circ f$ is a diagonal shop.*

Proof. It is sufficient to prove that $f \circ f^{-1}$ is diagonal. Let $a \in f \circ f^{-1}(b)$. Then there exists y s.t. $b \in f^{-1}(y)$ and $y \in f(a)$. Thus, $y \in f(b)$ and $a \in f^{-1}(y)$, i.e. $b \in f \circ f^{-1}(a)$.

Let f and g be diagonal shops on the domain D . The minimal diagonal shop h containing both f and g as a sub-shop is said to be the *join* of f and g . The *union* $(f \cup g)$ of f and g is the shop given by, for all $x \in D$, $(f \cup g)(x) := f(x) \cup g(x)$.

Lemma 3. *Let f and g be diagonal shops on the domain $[n]$. The join of f and g is $(f \cup g)^n = (f \circ g)^n = (g \circ f)^n$. In particular, DSMs are closed under join.*

Proof. Consider the union $f \cup g$. Since id_n is a sub-shop of both f and g , i.e. \mathcal{G}_f and \mathcal{G}_g are both reflexive, we can see $f \cup g \subseteq f \circ g \subseteq (f \cup g)^2$ and $f \cup g \subseteq g \circ f \subseteq (f \cup g)^2$. The join h of f and g contains exactly those $a \in h(b)$ and $b \in h(a)$ for which there is a path in $f \cup g$ from a to b (and b to a). By reflexivity of $f \cup g$ this is equivalent to there being an n -path between a and b in $(f \cup g)$, which is equivalent to there being an edge in $(f \cup g)^n$. Noting $(f \cup g)^n = (f \cup g)^{2n}$, the result follows.

It follows that there is a unique maximal diagonal shop in any DSM M , where the diagonals are ordered by sub-shops. Let g be a diagonal shop on the domain $[n]$ with associated partitions P_1, \dots, P_m . We say that a shop f *respects* g if **neither**

- (i) exists a, b and c, d s.t. a, b are in the same partition P_i and c, d are in distinct partitions P_j, P_k , respectively, and $c \in f(a)$ and $d \in f(b)$, **nor**
- (ii) exists a, b and c, d s.t. a, b are in distinct partitions P_j, P_k , respectively, and c, d are in the same partition P_i and $c \in f(a)$ and $d \in f(b)$.

Lemma 4. *If f does not respect the diagonal g , then either $(f \circ g) \circ (g \circ f^{-1})$ or $(f^{-1} \circ g) \circ (g \circ f)$ is a diagonal shop that is not a sub-shop of g .*

Proof. If f does not respect g because of Item (i) above, then $h := (f \circ g) \circ (f \circ g)^{-1}$ is diagonal s.t. $h(c) \supseteq \{c, d\}$. The result follows as $(f \circ g) \circ (f \circ g)^{-1} = (f \circ g) \circ (g \circ f^{-1})$.

If f does not respect g because of Item (ii) above, then f^{-1} does not respect g because of Item (i) above. The result follows.

Lemma 5. *If g is the maximal diagonal in a DSM M and f is a blurred permutation that respects g , then $f \in M$ iff there exists $f' \subseteq f$ s.t. $f' \in M$.*

Proof. The forward direction is trivial. The backward direction follows from the observation that $f \subseteq f' \circ g$, for any $f' \subseteq f$.

3.2 Automorphisms of \mathcal{F}_n

The lattice \mathcal{F}_n has a collection of very obvious automorphisms corresponding to the permutations of S_n , in which one transforms a DSM M to M' by the uniform relabelling of the elements of the domain according to some permutation. We will not dwell on these automorphisms other than to give the example that $M := \langle \begin{smallmatrix} 1|12 \\ 2|2 \\ 3|3 \end{smallmatrix} \rangle$ maps to $M' := \langle \begin{smallmatrix} 1|1 \\ 2|2 \\ 3|23 \end{smallmatrix} \rangle$ under the permutation $\{1 \mapsto 3, 2 \mapsto 2, 3 \mapsto 1\}$.

There is another, more interesting, automorphism of \mathcal{F}_n , which we will call the *inverse* automorphism. We do not close our DSMs under inverse because they were defined in order that the given Galois connections held. It is not hard to verify that if M is a DSM, then $\{f^{-1} : f \in M\}$ is also a DSM, which we call the *inverse* and denote M^{-1} . It is also easy to see that $f = (f^{-1})^{-1}$ and $M = (M^{-1})^{-1}$, from where it follows that inverse is an automorphism of \mathcal{F}_n .

3.3 Properties of Inverse

Call a structure \mathcal{B} *she-complementative* if $\text{shE}(\mathcal{B}) = \text{shE}(\mathcal{B})^{-1}$. Note that, if F is a DSM, then so is F^{-1} . In fact, this algebraic duality resonates with the de Morgan duality of \exists and \forall , and the complexity-theoretic duality of NP and co-NP [5].

Lemma 6. *For all \mathcal{B} , $\text{shE}(\mathcal{B}) = \text{shE}(\overline{\mathcal{B}})^{-1}$.*

Proof. It follows from the definition of she that f is a she of \mathcal{B} iff f^{-1} is a she of $\overline{\mathcal{B}}$.

We are now in a position to derive our main classification theorem.

Theorem 5. *A DSM N is a BPS iff $N = N^{-1}$.*

Proof. It is straightforward to see that a BPS N is s.t. $N = N^{-1}$. Specifically, if $f \in N$ then f is derived from a shop g of a permutation in a group M . The inverse f^{-1} may be derived in the same manner from the inverse g^{-1} of g .

Now suppose N is s.t. $N = N^{-1}$. Let g be the maximal diagonal shop in N . Let P_1, \dots, P_m be the associated partitions of g in the manner previously discussed. Let M be the blurred symmetric group formed from S_m by the partitions P_1, \dots, P_m . We claim $N \subseteq M$. This follows from Lemmas 3 and 4, since, if $N \not\subseteq M$, then some shop $f \in N$ fails to respect g , contradicting the maximality of the diagonal g . From Lemma 5 it follows that N contains a blurred permutation f of M iff it contains any sub-shop $f' \subseteq f$. From closure under inverse and sub-shops it follows that N must be a BPS.

We may now give a complexity classification for she-complementative structures based on the following result of [5].

Lemma 7 ([5]). *If $\text{shE}(\mathcal{B})$ is a BPS derived from S_m , for $m \geq 2$, then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) is Pspace-complete.*

Corollary 2. *If \mathcal{B} is she-complementative then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) is either in L or is Pspace-complete.*

Proof. We know from Theorem 5 that $\text{shE}(\mathcal{B})$ is a BPS. If it is a BPS formed from the trivial group S_1 , then $\text{shE}(\mathcal{B})$ contains all shops. It is easy to see that $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) is in L (indeed one may evaluate the quantified variables in an instance arbitrarily - for more details see [5]). If $\text{shE}(\mathcal{B})$ is a BPS formed from S_m , with $m \geq 2$, then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) is Pspace-complete by Lemma 7.

3.4 She-Complementative Classes

We now consider natural classes that are she-complementative. A *tournament* is a digraph \mathcal{H} s.t. for each $x, y \in H$ exactly one of (x, y) and (y, x) is in $E^{\mathcal{H}}$.

Lemma 8. *Tournaments are she-complementative.*

Proof. Let \mathcal{H} be a tournament and let $f \in \text{shE}(\mathcal{H})$. We will prove $f^{-1} \in \text{shE}(\mathcal{H})$. Suppose $y_1 \in f^{-1}(x_1)$ and $y_2 \in f^{-1}(x_2)$ and $(x_1, x_2) \in E^{\mathcal{H}}$; we must prove that $(y_1, y_2) \in E^{\mathcal{H}}$. The case $x_1 = x_2$ is not possible as \mathcal{H} has no self-loops. The case $x_1 \neq x_2$ but $y_1 = y_2$ is also not possible, since it implies $\{x_1, x_2\} \subseteq f(y_1)$ (by the pigeonhole principle there must be $y' \neq y''$ s.t. there is x' with $x' \in f(y')$ and $x' \in f(y'')$ – but \mathcal{H} has either $(y'', y') \in E^{\mathcal{H}}$ or $(y', y'') \in E^{\mathcal{H}}$ but no self-loop (x', x')). Finally, we may assume $x_1 \neq x_2$ and $y_1 \neq y_2$. One of the edges (y_1, y_2) or (y_2, y_1) is in $E^{\mathcal{H}}$, and it can not be (y_2, y_1) as this would force $(x_2, x_1) \in E^{\mathcal{H}}$ by the she f . The result follows.

Analysis of the proof of Lemma 8, together with Lemma 7, now gives us the following.

Corollary 3. *If \mathcal{H} is a tournament on at least two vertices, then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is Pspace-complete.*

We conclude this section with another example of a class of structures that are she-complementative. The proof of the following is a simple application of Lemma 6

Lemma 9. *If \mathcal{B} is such that, for each $R \in \mathcal{B}$ we also have the (set-theoretic) complement $\bar{R} \in \mathcal{B}$, then \mathcal{B} is she-complementative.*

Note that dichotomy of $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) when \mathcal{B} satisfies the condition of the previous lemma follows from the classification for $\{\neg, \exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) given in 6.

4 Digraph Templates

We now extend the result for tournaments of the previous section. A *semicomplete digraph* is an \mathcal{H} s.t. for each $x, y \in H$ at least one of (x, y) and (y, x) is in $E^{\mathcal{H}}$.

Lemma 10. *Let \mathcal{H} be a semicomplete digraph. Then $\text{shE}(\mathcal{H})$ is a permutation subgroup.*

Proof. We will prove that $\text{shE}(\overline{\mathcal{H}})$ is a permutation subgroup, whereupon the result will follow from Lemma 6 as $\text{shE}(\overline{\mathcal{H}})$ would be closed under inverse.

Since \mathcal{H} is semicomplete, $\overline{\mathcal{H}}$ is reflexive digraph with no induced clique of size > 1 . Consider $f \in \text{shE}(\overline{\mathcal{H}})$. For each $x \in H$, $|f(x)| = 1$ – for suppose $f(x) \supseteq \{y_1, y_2\}$ with $y_1 \neq y_2$, then the self-loop at x implies there is a double edge (y_1, y_2) and (y_2, y_1) , which is impossible as there is no induced clique of size > 1 . It follows that $\text{shE}(\overline{\mathcal{H}})$ is a permutation subgroup.

Lemma 7 now gives us the following.

Corollary 4. *If \mathcal{H} is a semicomplete digraph on at least two vertices, then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is Pspace-complete.*

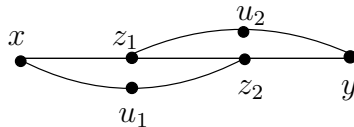
Recall that a graph is a digraph whose edge relation is symmetric (and may have self-loops). A graph \mathcal{H} is *trivial* if $E^{\mathcal{H}} = \emptyset$ or H^2 . An *isolated vertex* x in \mathcal{H} is s.t. $\forall y \in H (x, y), (y, x) \notin E^{\mathcal{H}}$. Conversely, a *dominating vertex* x in \mathcal{H} is s.t. $\forall y \in H (x, y), (y, x) \in E^{\mathcal{H}}$. A graph is *bipartite* if its vertices admit a partition into two sets s.t. all edges are strictly between the two partition sets. In a graph \mathcal{H} the distance between vertices x and y , $d(x, y)$, is the minimal length of a path that connects them (or infinity otherwise). It follows, under this definition, that the distance between a vertex and itself is at most two, unless that vertex is isolated. The *diameter* of a graph \mathcal{H} is the maximum value of $\{d(x, y) : x, y \in H\}$. We will use the following result from [7,6,5]

Lemma 11 ([7]). *If \mathcal{H} is a graph with an isolated vertex, some edge and no self-loops (respectively, dominating vertex, some non-edge and no loopless vertex) then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is NP-complete (respectively, co-NP-complete).*

Proof (Sketch). We sketch the argument for NP (that for co-NP is symmetric). Membership of NP comes from the fact that in the evaluation of a sentence ϕ of $\{\exists, \forall, \wedge, \vee\}$ -FO on \mathcal{H} one may evaluate all of the universal variables as the isolated vertex x . For completeness, one reduces from the NP-hard evaluation problem for existential positive fo logic without equality (see [6]).

Theorem 6. *If \mathcal{H} is a graph of diameter at least 3 then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is NP-hard.*

Proof. We will prove the result for \mathcal{H} of diameter 3 – the proof for larger diameters is similar. We will give a reduction from *not-all-equal 3-satisfiability* (see [10]). From an instance I of not-all-equal 3-sat we build a sentence ϕ of $\{\exists, \forall, \wedge, \vee\}$ -FO in the following manner. Firstly, we must find x, y s.t. $d(x, y) = 3$, so ϕ begins $\forall x, y E(x, y) \vee \exists z E(x, z) \wedge E(z, y) \dots$ Note that any x, y for which this is not already satisfied are s.t. $d(x, y) = 3$ (and such x, y exist). Now ϕ continues $\dots \exists z_1, z_2 E(x, z_1) \wedge E(z_1, z_2) \wedge E(z_2, y) \dots$ Define the unary relations $u \approx z_1$ to be $E(x, u) \wedge E(u, z_2)$ and $u \approx z_2$ to be $E(z_1, u) \wedge E(u, y)$. The crucial observation is that the sets $\{u : u \approx z_1\}$ and $\{u : u \approx z_2\}$ are necessarily *disjoint* and may be used to encode the true and false of not-all-equal 3-sat. In the following diagram $u_1 \approx z_1$ and $u_2 \approx z_2$.



For each variable V_i in the instance I we introduce $\exists v_i$ to the interior of ϕ . For each triple $(V_{a_1}, V_{a_2}, V_{a_3})$ constrained to be not-all-equal, we introduce the sequence $(v_{a_1} \approx z_1 \wedge v_{a_2} \approx z_1 \wedge v_{a_3} \approx z_2) \vee (v_{a_1} \approx z_1 \wedge v_{a_2} \approx z_2 \wedge v_{a_3} \approx z_1) \vee \dots$ etc., where we have all possible not-all-equal assignments. That I is satisfiable by a not-all-equal assignment iff ϕ is true on \mathcal{H} is clear by construction.

We note that this result is optimal in the sense that there is a graph \mathcal{H}_2 of diameter 2 that is unlikely to have $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}_2) being NP-hard, as $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}_2) is in co-NP (take \mathcal{H}_2 with vertices $\{1, 2, 3\}$ and edge set $\{(1, 2), (2, 1), (2, 2), (2, 3), (3, 2)\}$, and appeal to Lemma 11). The following is a conjecture about the complexity of QCSPs from [8].

Conjecture 1 ([8]). If \mathcal{H} is a connected non-bipartite graph then QCSP(\mathcal{H}) is Pspace-complete.

Theorem 7. *Let \mathcal{H} be an antireflexive graph or a reflexive graph, then*

- if \mathcal{H} is trivial, then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is in L.
- if \mathcal{H} is non-trivial and with an isolated vertex, then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is NP-complete.
- if \mathcal{H} is non-trivial and with a dominating vertex, then $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is co-NP-complete.

And, assuming Conjecture 1,

- otherwise, $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is Pspace-complete.

Proof. In the L case, shE(\mathcal{H}) contains all shops – it is easy to see that $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) is in L (evaluate the quantified variables in an instance arbitrarily – see [5]). The NP-complete and co-NP-complete cases follow from Lemma 11.

For the Pspace-complete case, we may reduce from QCSP(\mathcal{H}) – essentially a subproblem of $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) – if \mathcal{H} is antireflexive, connected and non-bipartite (appealing to Conjecture 1). We may do likewise if $\overline{\mathcal{H}}$ is such a graph. This is because $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{H}) and the complement of $\{\exists, \forall, \wedge, \vee\}$ -FO($\overline{\mathcal{H}}$) are polynomially equivalent (by de Morgan duality – $\mathcal{H} \models \phi$ iff $\overline{\mathcal{H}} \not\models \phi'$ where ϕ' is derived from ϕ by swapping all instances of \exists and \forall , and all instances of \wedge and \vee see [6]) and Pspace is closed under complementation. We are left with the cases where \mathcal{H} is antireflexive and bipartite or not connected (and their reflexive complements). In both of these cases we can $\{\exists, \forall, \wedge, \vee\}$ -FO define some \mathcal{H}' that is the union of $m \geq 2$ reflexive cliques (whereupon hardness follows from Lemma 7 as shE(\mathcal{H}') is a BPS derived from S_m). In the disconnected case, for $|H| = n$, we will define the edge relation in \mathcal{H}' to be given by being in the same connected component, vis

$$E(x, y) \vee \bigvee_{j=1}^n \exists w_1, \dots, w_j E(x, w_1) \wedge \dots \wedge E(w_j, y).$$

For the bipartite case we will just consider even length paths, defining the edge relation by

$$(\exists w_1 E(x, w_1) \wedge E(w_1, y)) \vee \bigvee_{j=1}^n \exists w_1, \dots, w_{2j+1} E(x, w_1) \wedge \dots \wedge E(w_{2j+1}, y).$$

Note that this provides all self-loops, as there are no isolated vertices in either of these cases.

5 Final Remarks

We have continued the study of the complexity of the problems $\{\exists, \forall, \wedge, \vee\}$ -FO(\mathcal{B}) through an analysis of the lattice \mathcal{F}_n .

It would be interesting to try to derive a generalisation of Corollary 2 for the case that $\text{shE}(\mathcal{B})$ does not necessarily equal, but is isomorphic to, $\text{shE}(\mathcal{B})^{-1}$. This includes several interesting new cases (see 9), but we would hope that dichotomy between L and Pspace-complete still holds.

It would also be interesting to improve Theorem 6 to Pspace-completeness. In fact, in all cases we know, one can rely on ad-hoc arguments to prove Pspace-hardness; but we know of no general mechanism for this result.

References

1. Börner, F.: Total multifunctions and relations. In: AAA60: Workshop on General Algebra, Dresden, Germany (2000)
2. Börner, F., Krokhin, A., Bulatov, A., Jeavons, P.: Quantified constraints and surjective polymorphisms. Tech. Rep. PRG-RR-02-11, Oxford University (2002)
3. Bulatov, A.A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM* 53 (1), 66–120 (2006)
4. Lynch, N.: Log space recognition and translation of parenthesis languages. *J. ACM* 24, 583–590 (1977)
5. Madelaine, F., Martin, B.: The complexity of positive first-order logic without equality. In: Symposium on Logic in Computer Science, pp. 429–438 (2009)
6. Martin, B.: First order model checking problems parameterized by the model. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) CiE 2008. LNCS, vol. 5028, pp. 417–427. Springer, Heidelberg (2008)
7. Martin, B.: Model checking positive equality-free FO: Boolean structures and digraphs of size three. CoRR abs/0808.0647 (2008)
8. Martin, B., Madelaine, F.: Towards a trichotomy for quantified H-coloring. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 342–352. Springer, Heidelberg (2006)
9. Martin, B., Martin, J.: The complexity of positive first-order logic without equality II: the four-element case. To appear CSL (2010)
10. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Reading (1994)
11. Schaefer, T.: The complexity of satisfiability problems. In: STOC (1978)

Constraint Based Scheduling to Deal with Uncertain Durations and Self-Timed Execution^{*}

Michele Lombardi and Michela Milano

DEIS, Università di Bologna V.le Risorgimento 2, 40136, Bologna, Italy

Abstract. In this paper we propose off-line and on-line extensions to the Resource Constrained Project Scheduling Problem. The off-line extension is a variant of RCPSP with time lags and uncertain, bounded activity durations. In this context we improve over our previous work presented in [12] by proposing an incremental flow computation for finding minimal conflict sets and a set of filtering rules for cumulative constraint propagation. The on-line extension is based instead on considering an on-line semantics such as the Self-Timed Execution and take it into account in the scheduling process. Adding the on-line aspect to the problem makes the CSP framework no longer suitable. We have extended the CSP framework to take into account general search decisions and auxiliary unbound variables. An extensive set of experimental results show an improvement of up to two orders of magnitude over our previous approach.

1 Introduction

In this paper we consider two notable extensions to the traditional Resource Constrained Project Scheduling Problem (RCPSP) that are widely used in practice. The classical RCPSP is defined on a set of activities that are linked via precedence relations and use a given amount of limited available cumulative resources. A solution of a RCPSP is an assignment of starting time to activities such that all temporal and resource constraints are satisfied and the completion time minimized.

Practically useful extensions to the RCPSP mainly concern two aspects: the off-line model and the on-line execution. Off-line extensions of the classical RCPSP include:

- time windows on the activity starting time,
- minimum and maximum time lags between the execution of two activities
- uncertain (bounded) durations: activity durations cannot be decided but range at run time between minimum and maximum known values.

In particular, uncertain durations make the extended problem no longer suitable for scheduling by assignment of starting times. Similarly to the Precedence

^{*} The work described in this publication was supported by the PREDATOR Project funded by the European Community's 7th Framework Programme, Contract FP7-ICT-216008 and by the ARTEMIS project SMECY, ARTEMIS-2009-1-100230.

Constraint Posting approach we hence assume the RCPSP solution to be an extended project graph with additional precedence constraints; this must be resource conflict free **for all combination of activity durations**. In the off-line setting the main contribution of this work are two improvements over our previous paper [12] and namely, the incremental flow computation for finding minimal conflict sets and a set of filtering rules for cumulative constraint propagation.

The on-line execution schema is also taken into account in this paper as a major contribution. The extended graph derived by the addition of new precedence constraints has some flexibility that can be exploited at run time. However, we can suppose that a run-time semantics makes some on-line executions infeasible. One widely used execution semantics in the field of system design is the Self-Timed Execution (STE); this policy requires that, if all predecessors of a task i have completed their execution, task i cannot be delayed. Additional precedence constraints (added to ensure the network is dynamically controllable and conflict-free) should obey STE restrictions as well.

Adding the on-line aspect to the problem complicates the overall picture and makes the CSP framework no longer suitable; the last important contribution of this paper is an extension of the CSP definition to take into account (1) search decisions not in the form of variable assignments and (2) auxiliary variables not necessarily bound in the final solution.

We present an extensive set of experimental results, showing the algorithm achieves up to two orders of magnitude speed ups w.r.t. the approach proposed in [12]. In addition, having considered the on-line execution schema has substantially reduced the number of precedence constraints added, therefore improving flexibility.

2 Problem Description

Constraint based scheduling stems out of the classical Resource Constrained Project Scheduling Problem (RCPSP), consisting in scheduling a set A of precedence connected activities a_i over a set R of limited capacity resources r_k . The structure of the precedence relations is usually captured in the RCPSP via a directed graph $G = \langle A, E \rangle$ (*project graph*), where E is the set of arcs (a_i, a_j) . Each activity requires an amount rq_{ik} of each resource $r_k \in R$ (where $rq_{ik} = 0$ to denote no requirement) and c_k is the capacity of r_k . Resources are renewable (i.e. the consumed capacity is released when an activity is over) and activities are assumed to have fixed durations (e.g. d_i); a schedule is defined as an assignment of start times to activities and a common RCPSP objective is to minimize the project makespan; resource capacity cannot be exceeded at any point of time. Figure 1A shows a project graph, with all activities requiring different amounts (reported next to the nodes) of a single resource with capacity 3.

In CP based scheduling activities are usually modeled by pairs of finite, integer variables S_i, E_i , respectively representing the start/end of activity a_i . Start/end variables must satisfy $E_i = S_i + d_i$. Precedence relations (denoted as $a_i \prec a_j$) can be modeled as simple inequalities $E_i \leq S_j$; unlike in the basic RCPSP, time

windows on the activity execution, specified via release times rs_i and deadlines dl_i are naturally taken into account in CP by posting $S_i \geq rs_i$ and $E_i \leq dl_i$. Resource over-usages are prevented by means of the cumulative constraint, enforcing:

$$\sum_{S_i \leq t < E_i} r_{q_{ik}} \leq c_k \quad \forall k, \forall t = 0, \dots, eoh \tag{1}$$

Where eoh (End Of Horizon) is the same as the maximum deadline (i.e. $eoh = \max_i(dl_i)$).

Finally, minimum/maximum time lags on the precedence relations $(\delta_{ij}/\Delta_{ij})$ can be easily taken into account in CP by replacing the simple inequalities $E_i \leq S_j$ with $\delta_{ij} \leq E_i - S_j \leq \Delta_{ij}$. As in the basic RCPSP, a schedule consists into an assignment of start times to activities.

2.1 Uncertain, Bounded Durations

Here, as in the foregoing work [12] we assume activities have *uncertain duration*, bounded by a minimum value d_i and a maximum D_i . Unlike *variable durations*, which can be decided at search time and may depend on other problem variables, uncertain durations can only be observed. Here, we require that temporal and resource constraints are met for every possible execution scenario: this is a mandatory requirement whenever hard temporal constraints are to be (e.g. in a real time scenario).

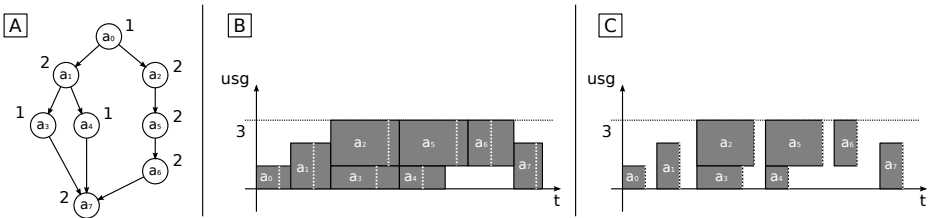


Fig. 1. A) A project network; B) A sample fixed-time schedule; C) Idle time appearing if durations are shorter

Even with uncertain durations, a feasible schedule can be provided by assuming worst case durations and fixing start times, as depicted in Figure 1B. However, if at run time some duration turns out to be shorter, then maximum time lags may be violated and a lot of idle time may be inserted (Figure 1C).

Ideally, one would like to be able to anticipate activities in case they last shorter. This is tackled in Precedence Constraint Posting approaches by assuming the presence of a simple *on-line dispatcher* with the ability to assign an actual start time to each a_i ; the dispatcher proceeds by choosing a start in the time window of S_i , compatible with the completion time of its predecessors a_j . In this

setting, a solution to the scheduling problem is a set set E' of additional precedence constraints, such that (A) the dispatcher always produces a time feasible schedule (i.e. dynamic controllability [17] holds) and (B) no resource conflict can arise at run-time.

Condition (B) holds if and only if the graph contains no Conflict Set, i.e. no set of possibly overlapping activities collectively overusing a resource. Formally:

Definition 1. *A Conflict Set is a set CS of activities such that:*

1. $\exists r_k \in R : \sum_{a_i \in CS} r_{q_{ik}} > c_k$
2. $\forall a_i, a_j \in CS, a_i \neq a_j : a_i \preceq a_j$ is consistent with the temporal constraints.

The decision version of the resulting scheduling problem consist in finding such a set E' of additional precedence constraints. A related optimality problem is that of determining the tightest global deadline eah for which such a set E' can be provided. The number of added precedence relations $|E'|$ can be considered a second cost metrics, as it impacts the flexibility of the provided solution, i.e. the amount of computation time which can be saved if activity durations are not maximum at run time.

2.2 Run-Time Semantics

The depicted setting, with an on-line dispatcher assigning the actual start time, is often the exact description of how the schedule is executed in practice. Moreover, in some cases such a dispatcher may be subject to further restrictions, collectively referred to as *run-time semantics*.

One of the main contribution of this paper is an extension of the outlined scheduling model to incorporate run-time semantics; as those are basically on-line enforced constraint, the integration of such an element is not straightforward. In particular, we focus here on dispatchers restricted to follow the so-called *Self Timed Execution* (STE dispatchers); STE forbids to delay an activity at run-time once all its predecessors are over.

In the following, we also refer to the dispatcher assumed by usual PCP approaches as NULL dispatcher (as it is subject to no additional restriction); this satisfies two important theorems:

Theorem 1. *The Conflict Sets of the input graph G in a PCP approach relying on the NULL dispatcher are the same as in a conventional scheduling approach where start times are fixed.*

Theorem 2. *By adding a new precedence constraint, we can only remove some of the Conflict Sets of the input graph G with the NULL dispatcher.*

Proofs are omitted due to lack of space. Note Theorem 1 does not hold in the general case. In particular, with STE:

1. the CS in G are a subset of those with the NULL Dispatcher (hence Theorem 1 does not hold);

2. by adding a new precedence constraint, we can both add and remove CS from G (hence Theorem 2 does not hold).

Basically, the restrictions enforced by STE avoid all CS which could result from delaying an activity; however, the addition of a precedence relation may “push” an activity along the timeline until new overlaps (and thus new conflicts) arise. This is in principle a major difference with conventional CP problems, where possible infeasibilities can only be removed as search deepens.

3 Related Work

The Resource Constrained Project Scheduling Problem has been extensively studied both in the CP and in Operations Research communities.

In particular, most of the RCPSP related work comes from an OR background; the full literature body is too large to fit the available space, hence only some key references are given. Work [4] represents an excellent (although a little dated) survey and proposes a widely popular notation. For more recent surveys, the reader may refer to [6], or to [9] for heuristic solution approaches.

Minimum and maximum time lags have been considered in the context of RCRPSP problem with fixed activity durations only; for a list of recent approaches one may refer to [6]. Uncertain durations are traditionally taken into account by Stochastic RCPSP (see [7] for a survey and [15] for a seminal work). Here the focus on expected makespan minimization; to the best of the author knowledge, deadlines and maximum time lag constraints have never been taken into account in the context of Stochastic RCPSP. For a broader discussion of scheduling techniques in the presence of uncertainty, see [2].

On the CP side, the book [1] stands as a main reference, while [10,3] (among the others) propose more advanced filtering techniques for the `cumulative` constraint.

The Precedence Constraint Posting approach is best formalized in [14], but has been extensively used in the past (see [11] and [15] as examples). The idea of branching over Conflict Sets was in fact introduced in [8] (in 1983).

Dynamic Controllability was introduced in [17] over Simple Temporal Networks with Uncertainty, for which controllability is achievable in polynomial time (see [13]). Finally, observe that the simple on-line dispatcher taken into account by our approach has nothing to do with the advanced on-line techniques used in [16]: in our approach, all computationally expensive optimization is performed prior to execution.

4 A Brief Revisitation of CSPs

The outlined scheduling setting presents unique features hardly fitting the usual CSP definition; on one side, solving a problem does not require all variables to be assigned (rather, all possible resource conflicts must be wiped out); on the

other, the lack of validity of Theorem 2 seems to compromise the underlying principles of CP depth first search.

We observe that similar issues are often encountered in practice, and equally often solved by seamlessly diverging from the usual CSP framework. We argue that such oddities rather point out the lack in the CSP definition of some important elements we all are familiar with. As a solution, we introduce, in the context of the tackled scheduling problem, a so-called Extended CSP (ECSP).

Definition 2. *An ECSP is a 4-tuple $\langle X, \mathcal{D}, C, T \rangle$ where $X = \{X_i\}$ is the set of variables, $\mathcal{D} = \{\mathcal{D}_i\}$ is the set of corresponding domains and $C = \{C_j\}$ is a set of constraints; $T = \{\tau_k\}$ is second set of constraints representing the possible decisions.*

In a nutshell, the following main differences with a conventional CSP exist and will be exemplified on the problem at hand:

#1: We assume problem constraints to embed a required consistency level; in detail, a constraint C_j does not list the allowed tuples, but rather allowed domain states. For example, suppose we have $X_0, X_1 \in [1..3]$ and a constraint $X_0 < X_1$ with required GAC. Then the constraint is satisfied when domains (D_1, D_2) are $([1], [2]), ([1], [3]), ([2], [3])$ as well as $([1], [2..3]), ([1..2], [3]), ([1..2], [2..3])$.

#2: A solution of a ECSP is a subset of decisions $T' \subseteq T$ such that $\langle X, \mathcal{D}, C \cup T', \emptyset \rangle$ is consistent. This accommodates many practical situations where there are auxiliary variables which may not need to be instantiated to get a solution. Search proceeds by moving decisions from the initial set T to C .

#3: Filtering is decision-aware. Formally, filtering can remove any value v in a domain which is not consistent with any subset T' of possible decisions ($T' \in 2^T$). This has an important consequence: checking whether a search node is a solution amounts to temporarily assume no more decisions can be taken (i.e. $T = \emptyset$) and testing consistency; now, as filtering is decision-aware, *assuming* $T = \emptyset$ may allow *additional propagation* and make consistent a previously inconsistent state.

5 The Proposed Scheduling Framework

Specifying a scheduling framework amounts to show how each of the elements presented in Section 2 can be modeled and tackled in the $\langle X, \mathcal{D}, C, T \rangle$ tuple. Precedence Constraint Posting and MCS based search serve as a basis for our model and our solution method.

5.1 Variables, Domains and Temporal Constraints

To represent activities and their relations over time, we rely on the temporal model we presented in [12], closely related to Simple Temporal Networks with Uncertainty (STNU, [17]). The model provides the following building blocks:

- **Time event variables** (T_i), for which a time window is specified; for each T_i , the domain \mathcal{D}_i is the time window itself.
- **Free constraints** ($T_i \xrightarrow{[d_{ij}, D_{ij}]} T_j$), meaning that *at least* a value d' in the interval $[d_{ij}, D_{ij}]$ must exist such that $T_j = T_i + d'$; $0 \leq d_{ij} \leq D_{ij}$ must hold.
- **Contingent constraints** ($T_i \xrightarrow{[d_{ij}; D_{ij}]} T_j$), meaning that T_i and T_j must have enough flexibility to allow $T_j = T_i + d'$ for each value $d' \in [d_{ij}, D_{ij}]$; $0 \leq d_{ij} \leq D_{ij}$ must hold.

The required consistency level is Generalized Arc Consistency, which is equivalent in this case to dynamic controllability. GAC can be enforced efficiently on the temporal network by means of linear cost propagations (see [12]). Figure 2 shows a very simple temporal network featuring four event variables, two contingent constraints and a free constraint.

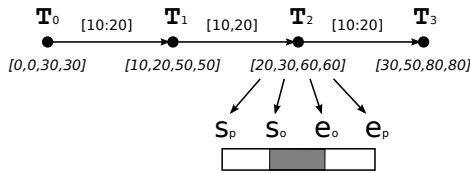


Fig. 2. The 4 value time window of time event variables

The time window of each event variable T_i is specified by means of 4 time values, namely $s_p(T_i)$, $s_o(T_i)$, $e_o(T_i)$, $e_p(T_i)$. Values $s_p(T_i)$ and $e_p(T_i)$ delimit the so-called *possible span* and specify the time span where the event may take place at run time; formally, let Σ be the set of possible execution scenarios σ (i.e. combination of task durations):

$$\forall t \in [s_p(T_i), e_p(T_i)], \exists \sigma \in \Sigma \text{ such that event "i" occurs at time } t$$

Values $s_o(T_i)$ and $e_o(T_i)$ bound the so-called *obligatory span*; if an event is forced to occur out of its *obligatory span* (i.e. if $s_p > e_o$ or $e_p < s_o$) dynamic controllability is compromised. Formally:

$$\forall \sigma \in \Sigma, \text{ event "i" occurs in }]0, s_o(T_i)[\cup]e_o(T_i), eoh[\Rightarrow \text{no dyn. controllability}$$

The 4 points time window allows one to test constant time whether a contingent or free constraint can be consistently added with the current CSP state.

Each activity a_i is modeled by introducing two event variables (referred to as S_i and E_i from now on) to represent its start/end time. Pairs of start/end variables related to the same activity are linked by a contingent constraint, while free constraint are used to model precedence relations; overall we have:

$$S_i \in [rs_i..eoh], E_i \in [0..dl_i] \quad \forall a_i \in A \tag{2}$$

$$S_i \xrightarrow{[d_i; D_i]} E_i \quad \forall a_i \in A \tag{3}$$

$$E_i \xrightarrow{[\delta_{ij}, \Delta_{ij}]} S_j \quad \forall (a_i, a_j) \in E \tag{4}$$

where we recall that d_i, D_i respectively are the minimum and maximum duration of activity a_i , while δ_{ij}, Δ_{ij} are the minimum and maximum time lag associated to arc (a_i, a_j) ; rs_i is the release time of a_i and dl_i the corresponding deadline. Figure 3 shows an outline of the temporal model corresponding to the project graph in Figure 1A/3A; contingent constraints are represented as solid arcs, while dashed arcs represent free constraints.

5.2 Resource Constraints

Renewable resources, such as those described in Section 2 are modeled by means of the cumulative global constraint, extended in order to deal with time event variables and uncertain durations. The required consistency level is achieved if and only if the time network is resource conflict free. Both the extended cumulative filtering and the consistency check procedure are described in the followings.

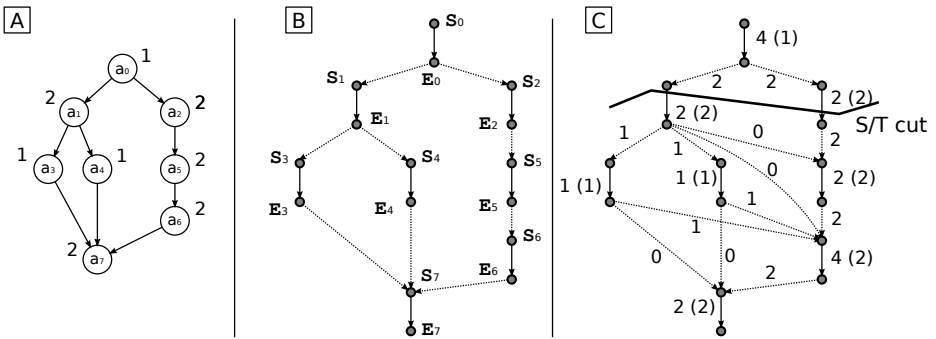


Fig. 3. A) The reference project Graph B) An outline of the corresponding temporal model C) Solving a minimum flow problem over the temporal network (minimum flow requirements are between round brackets)

Consistency Check: Checking consistency for cumulative constraint basically consists in detecting possible resource conflicts. We recall (see Definition 1) that a Conflict Set CS is a set of activities a_i such that:

1. $\forall a_i, a_j \in CS, a_i \neq a_j : a_i \preceq a_j$ and $a_j \preceq a_i$ are consistent with the current state of the CSP.
2. $\exists r_k \in R : \sum_{a_i \in CS} rq_{ik} > c_k$

In the context of the current time model, $a_i \preceq a_j$ translates to $E_i \xrightarrow{[0, eoh]} S_j$; the consistency of the constraint $a_i \preceq a_j$ can be checked in constant time (thanks to the four-point time window).

Note that, due to property (1), a Conflict Set is always a stable set on the project graph $G = \langle A, E \rangle$; as G is transitive and acyclic, its maximum weight stable set can be found in polynomial time by solving a minimum flow problem (see [5]). By weighting activities with their resource requirement, if we found that

the maximum weight stable set does not exceed the capacity, then the current CSP contains no conflict.

In practice (see [12]) the process is performed by i) selecting a target resource r_k , ii) annotating arcs in the temporal network corresponding to activities with their rq_{ik} values as minimum flow requirements; then by iii) augmenting with so-called *discovered* precedence constraints (i.e. pairs T_i, T_j such that $e_p(T_i) \leq_p(T_j)$). Finally, the minimum flow is computed via the (inverse) Edmond-Karp algorithm (an embodiment of the Ford-Fulkerson method); the S/T cut identifies the maximum weight stable set.

Note that, during the consistency check, the set of search decision taken so far is considered final ($T = \emptyset$ in Section 4); this may enable additional propagation due to STE constraints (see forthcoming Section 5.3) and lead to the discovery of more precedence constraints. For example, Figure 3C shows the minimum flow for the graph in Figure 3A; since STE does not allow activities do be delayed, assuming no more ordering decision can be taken lead to the discovery of $E_3 \rightarrow S_6$, $E_4 \rightarrow S_6$, $E_1 \rightarrow S_6$, $E_1 \rightarrow S_5$.

The chosen algorithm requires to start from a feasible flow; this should be as tight as possible, as its value directly impacts the complexity of Edmond-Karp (this is $O(|A| \cdot F)$, where F is the value of the feasible flow). In this work, we improved the method by using the minimum flow computed at a search node n' as feasible flow for each child node n'' .

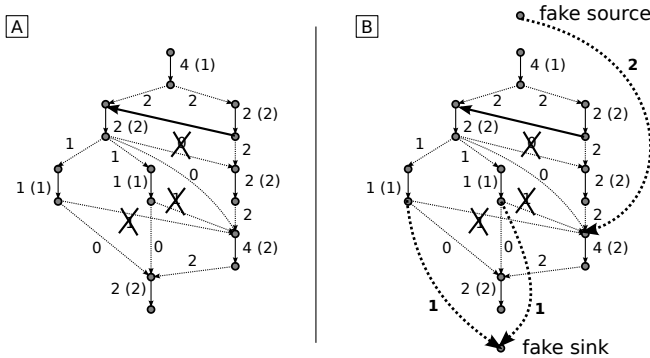


Fig. 4. A) Minimum flow at a search node, disrupted by the addition of a precedence relation B) Repaired flow

Note however that, since (with STE semantic) scheduling decisions may invalidate some previously discovered arcs, a flow-fixing phase must precede each minimization. In detail, this is done by introducing a fake source and fake sink event in the network, requiring 0 unit of each resource and having free constraint respectively to/from every event in the network. Then, for each arc (t_i, t_j) (with flow value $f > 0$) at node n , no longer present at node n' , we:

- route f flow units from the source to t_j and
- route f flow units from t_i to the sink.

The fixing procedure is completed in $O(n^2)$ and provides a feasible flow for n' , usually having pretty good quality. Figure 4A shows the minimum flow, disrupted by the addition of a precedence constraint $a_2 \preceq a_1$; Figure 4B depicts the corresponding repaired flow.

Extended Cumulative Filtering: Classical Propagation techniques for the cumulative constraint (such as edge finder [1], or balance [10]) can be used with uncertain, bounded durations by associating contingent constraints (introduced to model the activities in the project graph) to a pair of standard CP activities.

More in detail, the *possible span* of S_i is used as the domain for the start variable of a so-called *minimal* CP activity, having variable (as opposed to uncertain) duration in the interval $[d_i, D_i]$. The start of a *maximal* CP activity, with fixed duration equal to D_i is instead attached to the *obligatory span*. Note this technique in principle enables weaker results compared to a the use of ad-hoc filtering with uncertain durations, but allows one to easily leverage all the propagation techniques provided by state of the art CP solvers.

5.3 Handling STE and Global Consistency Check

Taking into account Self Timed Execution Semantic is the main contribution of this paper. In detail, that can be done by requiring each activity to start when the last of its explicit predecessors ends. Formally, let $A^+(a_i)$ be the set of activities such that $E_j \xrightarrow{[\delta_{ij}, \Delta_{ij}]} S_j$; then the following constraint must be satisfied by the on-line start time assignment and included in the C set:

$$S_i = \begin{cases} 0 & \text{if } |A^+(a_i)| = 0 \\ \max_{a_i \in A^+(a_i)} (E_j + \delta_{ij}) & \text{otherwise} \end{cases} \tag{5}$$

Note that the maximum time lag Δ_{ij} does not appear in the STE fundamental constraint. The required consistency level for the STE constraints is GAC.

Unfortunately, (5) is a conjunctive constraint when propagating from time 0 to eah , but a *disjunctive* constraint during propagation from time eah to 0. Due to the disjunctive part, the constraint polytope is not convex and GAC cannot be enforced in polynomial time in general.

As an exception, polynomial time GAC enforcement becomes feasible when, during the consistency check, T is set to \emptyset (i.e., the current decision set is marked as final). This allows STE constraint to perform a much stronger propagation during the consistency check. In such a situation STE propagation enforces:

$$s_p(S_i) = \begin{cases} 0 & \text{if } |A^+(a_i)| = 0 \\ \max_{a_i \in A^+(a_i)} (s_p(E_j) + \delta_{ij}) & \text{otherwise} \end{cases} \tag{6}$$

$$s_o(S_i) = \begin{cases} 0 & \text{if } |A^+(a_i)| = 0 \\ \max_{a_i \in A^+(a_i)} (s_o(E_j) + \delta_{ij}) & \text{otherwise} \end{cases} \tag{7}$$

$$e_o(S_i) = e_p(S_i) = s_o(S_i) \tag{8}$$

In other words, every activity is moved as close as possible to the schedule origin. Filtering rule [8](#), in particular, strongly reduces the e_p value of S_i variables (and consequently that of E_i). This may produce a large number of discoverable precedence relations (i.e. when $e_p(E_i) \leq s_p(E_i)$) which must be taken into account when checking the consistency of cumulative constraints (see Section [5.2](#)). Note the effect of the additional propagation must be undone when moving to the next search node.

5.4 Possible Decisions

The purpose of decision constraints is to solve inconsistencies in $\langle X, \mathcal{D}, C, \emptyset \rangle$. Essentially:

- some resource conflict may still exist (resource infeasibility)
- resource propagation and time window constraint may require an activity to be delayed and thus violate STE (STE infeasibility)

Therefore, decision constraints must be defined to fix both the situations; in detail, we introduce Conflict and STE Resolvers.

Conflict Resolvers: A resolver for conflict set CS is a free precedence relation $E_i \xrightarrow{[0, eoh]} S_j$ with $a_i, a_j \in CS$, posted in order to reduce the overall resource consumption of CS . If the conflict set is *minimal* (i.e. such that by removing any a_i from CS no resource over-usage occurs), then posting a single resolver wipes out the whole conflict.

STE Resolvers: An STE resolver for an infeasibility on a_i consists in choosing an arbitrary activity a_j as its last explicit predecessor. Once the activity is chosen, the resolver consists in the constraint $E_j \xrightarrow{[\delta_{i j}, \delta_{i j}]} S_i$, where $\delta_{i j}$ is assumed to be 0 if a_j is not an explicit predecessor of a_i ($a_j \notin A^+(a_i)$).

5.5 The Search Process

Once variables, domains, constraints (with the required consistency levels) and decisions are given, specifying the search process becomes a much simpler task. In particular, we adopt for the provided scheduling framework a Depth First Search method.

As in any CP approach, the computation at each node of the search tree starts by performing propagation, then proceeds with the consistency check: this involves (temporary) propagating STE constraints and running min-flow based conflict detection. If neither resource nor STE infeasibilities are reported, a solution has been hit, otherwise a choice point is opened. The actual behavior depends on the type of the detected infeasibility; priority is given to resource infeasibilities if both types are detected.

Solving Resource Infeasibilities: if a Conflict Set CS is identified on a resource r_k , the search proceeds by extracting an MCS from CS by applying greedy minimization; this consists in iteratively removing from CS the activity yielding the best reduced conflict set. Note that the additional propagation in the consistency check rules out Conflict Sets with no chance to occur, given the current set of ordering decisions, thus avoiding the insertion of useless constraints.

The quality of a CS is assessed as in [11] as an estimate of the preserved search space. In detail, preserved *possible span* and *obligatory span* must be distinguished; MCS with lower preserved obligatory span are preferred, while the preserved possible span is used to break ties.

The choice point in this case has a branch for each resolver compatible with the current CSP state; resolvers are then ranked by highest preserved obligatory span (the preserved possible span is used to break ties); in particular, let $\rho_0, \rho_1 \dots$ be the ranked resolvers. Along each branch k , resolvers $\rho_0, \dots, \rho_{k-1}$ are *forbidden* (as the corresponding branches have already been explored) and resolver ρ_k is posted. Forbidding a resolver $E_i \xrightarrow{[0, eoh]} S_j$ amounts to posting $S_j \xrightarrow{[1, eoh]} E_i$.

Solving STE Infeasibilities: If STE infeasibilities are detected on multiple activities a_i , the one with the lowest $s_p(S_i)$ is chosen; moreover.

Then all resolvers compatible with the current CSP state are identified and ranked according to the gap between a_i and the candidate predecessor a_j . Preference is accorded to resolvers between a_j and a_i having the smallest absolute difference $|s_o(E_j) - s_o(S_i)|$ (while $|s_p(E_j) - s_p(S_i)|$ is used to break ties). Finally, a choice point is opened with a branch for every resolver ρ_k ; unlike CS resolvers, in the current implementation STE resolvers are not forbidden on the right most branches.

6 Experimental Results

The proposed framework was implemented on top of IBM/ILOG Solver and Scheduler 6.7. The approach was tested on a RCPSp problem with uncertain durations and Self Time Execution semantic, arising in the design flow of real-time embedded systems. In detail, we considered four benchmarks sets of instances¹ (the same used in [12]), corresponding to two original groups of software application, mapped to two different platform.

Both original application groups were synthetically obtained through a specific instance generator, designed to mimic structure and features of real-world programs. Applications in group one have constant branching factor (number of successors of the activity starting a parallel session), ranging from 3 to 5, and scale in the number of software tasks; each software task can correspond to one or more activities, depending on the platform mapping. Applications in group 2 have fixed number of tasks (40) and scale along the branching factor, ranging from 2/4 to 6/8, exposing increasing parallelism.

¹ Available at www.lia.deis.unibo.it/Staff/MicheleLombardi/

We solved the scheduling problem in its optimality version, where the objective is to find the tightest deadline for which guarantees can be given (and the corresponding schedule). This was solved as a sequence of feasibility problem by applying binary search; the main advantage of the approach is to provide a lower bound on the tightest deadline as well as a feasible solution. All experiments were performed on a Core2 T7200, 2GHz with a time limit of 900 seconds. The solver described in this paper is compared with the one presented in [12].

Table 1. Results for application group 1

	size	nodes	arcs	STE solver					Previous MinFlow solver				
				<i>nprec</i>	<i>time</i>	#MCS	<i>tmcs/time</i>	> <i>TL</i> lb/ub	<i>nprec</i>	<i>time</i>	#MCS	<i>tmcs/time</i>	> <i>TL</i> lb/ub
Platform A	20	41/49	50/62	14	0.21(0.04)	256	0.08	0	19	0.46(0.18)	222	0.94	0
	30	56/66	67/85	20	0.37(0.06)	523	0.25	0	28	91.2(284.72)	29373	0.95	1 0.83
	40	75/82	93/105	24	1.33(2.68)	2060	0.19	0	37	3.5(1.46)	394	0.98	0
	50	93/103	115/133	38	0.96(0.62)	1345	0.38	0	57	189.49(375.91)	18533	0.98	2 0.84
	60	110/119	138/155	47	7.09(14.9)	5779	0.78	0	74	554.32(461.47)	27302	0.98	6 0.95
	20	29/36	38/49	7	0.15(0.02)	109	0.03	0	13	0.13(0.06)	105	0.86	0
Platform B	30	41/52	52/71	9	0.26(0.05)	127	0.11	0	15	0.45(0.45)	175	0.86	0
	40	54/60	72/82	10	0.37(0.05)	137	0.11	0	17	0.58(0.3)	153	0.88	0
	50	65/78	89/108	17	0.57(0.08)	226	0.12	0	27	91.9(284.28)	11522	0.95	1 0.99
	60	78/86	106/122	25	0.75(0.12)	446	0.11	0	37	93.31(283.63)	11711	0.97	1 0.98

Table 1 and 2 respectively show results for the first and the second set of original applications, mapped to the two considered hardware platforms. In detail, platform A features 16 unary resource (representing processors) and 32 cumulative resources (representing point to point communication channels). Platform B has 4 cumulative resource representing multi-core clusters and 8 to model communication channels. Each row summarizes results for a group of 10 instances; column tells the number of tasks before mapping, while *bfactor* in Table 2 specifies the branching factor for the row. Columns *nodes* and *arcs* tell the minimum/maximum number of arcs after the mapping (i.e. the actual input for the scheduling process). For both solvers, *nprec* reports the number of additional precedence constraint, while *time* is the average solution time (with standard deviation between round brackets). Column #MCS reports the average number of considered Minimal Conflict Set, *tmcs/time* is the ration between the total solution time and the time spend in MCS detection; finally > *TL* tells the number of timed out instances and *lb/ub* the ratio between the lower bound and an upper bound on the tightest achievable global deadline, computed by binary search.

One can see how the new solver obtains up to two order of magnitude speed-ups compared to the previously presented one; this is mainly due to the introduced support for standard CP propagation (namely, timetable + balance) algorithms and of the use of STE semantic to focus on MCS with an actual chance to occur with the currently posted precedence constraints. A second relevant difference is the relative amount of time spent in MCS detection, which

Table 2. Results for application group 2

	bfactor	nodes	arcs	STE solver					Previous MinFlow solver						
				nprec	time	#MCS	tmcs/time	> TL	lb/ub	nprec	time	#MCS	tmcs/time	> TL	lb/ub
Platform A	2/4	74/84	90/108	22	0.6(0.11)	437	0.13	0	34	121.21(288.9)	16173	0.96	1	0.97	
	3/5	78/84	98/109	30	0.68(0.1)	701	0.22	0	45	192.79(374.98)	26066	0.97	2	0.9	
	4/6	78/88	99/115	38	0.81(0.17)	1107	0.61	0	55	534.6(455.86)	78117	0.95	5	0.83	
	5/7	79/90	100/119	38	90.32(282.55)	1189	0.55	1	0.97	54	184.61(377.86)	18452	0.98	2	0.5
	6/8	83/95	107/126	47	2.86(5.8)	5198	0.9	0	69	633.74(430.84)	89446	0.96	7	0.88	
Platform B	2/4	55/62	73/86	9	0.4(0.05)	124	0.05	0	16	0.67(0.66)	169	0.87	0		
	2/4	55/66	75/88	14	0.43(0.04)	270	0.21	0	27	91.16(284.59)	17969	0.91	1	0.94	
	2/4	57/69	78/95	19	0.44(0.06)	289	0.19	0	34	94.11(283.65)	20649	0.95	1	0.98	
	2/4	52/72	73/101	21	0.45(0.11)	403	0.35	0	32	91.92(284.13)	5954	0.96	1	0.83	
	2/4	58/67	84/98	25	0.47(0.07)	521	0.26	0	39	181.96(378.89)	35361		2	0.8	

takes the largest portion of the search time for the previous solver, but is considerably lower for the new solver. Once again, this is a combined effect, due to the much larger time required by advanced propagation and the faster MCS detection enabled by incremental flow correction. Finally, taking into account Self Timed Execution enabled the solver to obtain feasible schedules by posting consistently fewer precedence constraint; this is a significant result in the context of scheduling for Embedded System Design, since all additional precedence constraints have to be implemented by means of actual changes to the original application and their number should be ideally minimized.

7 Conclusion

An efficient complete solver for facing a significant variant of Resource Constraint Project Scheduling with minimum and maximum time lags and variable durations is proposed. The concept of durations here is particularly challenging as we have to consider constraint feasibility for each possible activity duration combination. In addition, we propose a notable extension of the traditional off-line RCPSP that takes into account the Self-Timed Execution semantics and ensures dynamic controllability and resource-conflict free solutions.

References

1. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling. Kluwer Academic Publishers, Dordrecht (2001)
2. Beck, J.C., Davenport, A.J.: A survey of techniques for scheduling with uncertainty (2002), <http://www.eil.utoronto.ca/profiles/chris/gz/uncertainty-survey.ps>
3. Beldiceanu, N., Poder, E.: A Continuous Multi-resources umulative Constraint with Positive-Negative Resource Consumption-Production. In: Van Hentenryck, P., Wolsey, L.A. (eds.) CPAIOR 2007. LNCS, vol. 4510, pp. 214–228. Springer, Heidelberg (2007)

4. Brucker, P., Drexel, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112(1), 3–41 (1999)
5. Golombic, M.: *Algorithmic Graph Theory And Perfect Graphs*, 2nd edn. Elsevier, Amsterdam (2004)
6. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* (2009)
7. Herroelen, W., Leus, R.: Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research* 165(2), 289–306 (2005)
8. Igelmund, G., Radermacher, F.J.: Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks* 13(1), 1–28 (1983)
9. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research* 174(1), 23–37 (2006)
10. Laborie, P.: Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artif. Intell.* 143(2), 151–188 (2003)
11. Laborie, P.: Complete MCS-Based Search: Application to Resource Constrained Project Scheduling. In: Proc. of IJCAI, pp. 181–186. Professional Book Center (2005)
12. Lombardi, M., Milano, M.: A Precedence Constraint Posting Approach for the RCPSP with Time Lags and Variable Durations. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 569–583. Springer, Heidelberg (2009)
13. Morris, P.H., Muscettola, N., Vidal, T.: Dynamic control of plans with temporal uncertainty. In: Proc. of IJCAI, pp. 494–502 (2001)
14. Policella, N., Cesta, A., Oddi, A., Smith, S.F.: From precedence constraint posting to partial order schedules: A CSP approach to Robust Scheduling. *AI Commun.* 20(3), 163–180 (2007)
15. Stork, F.: Stochastic resource-constrained project scheduling. PhD thesis, Technische Universität Berlin (2001)
16. Van Hentenryck, P., Bent, R., Mercier, L., Vergados, Y.: Online stochastic reservation systems. *Annals of Operations Research* 171(1), 101–126 (2009)
17. Vidal, T., Fargier, H.: Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.* 11(1), 23–45 (1999)

Local Consistency and SAT-Solvers

Justyna Petke and Peter Jeavons

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK
{justyna.petke,Peter.Jeavons}@comlab.ox.ac.uk

Abstract. In this paper we show that the power of using k -consistency techniques in a constraint problem is precisely captured by using a particular inference rule, which we call positive-hyper-resolution, on the direct Boolean encoding of the CSP instance. We also show that current clause-learning SAT-solvers will deduce any positive-hyper-resolvent of a fixed size from a given set of clauses in polynomial expected time. We combine these two results to show that, without being explicitly designed to do so, current clause-learning SAT-solvers efficiently simulate k -consistency techniques, for all values of k . We then give some experimental results to show that this feature allows clause-learning SAT-solvers to efficiently solve certain families of CSP instances which are challenging for conventional CP solvers.

1 Introduction

One of the oldest and most central ideas in constraint programming, going right back to Montanari's original paper in 1974 [22], is the idea of using *local consistency* techniques to prune the search space [11]. The idea of arc-consistency was introduced in [21], and generalised to k -consistency in [16]. Modern constraint solvers generally employ specialised propagators to prune the domains of variables to achieve some form of generalised arc-consistency, but do *not* attempt to enforce higher levels of consistency, such as path-consistency.

By contrast, the software tools developed to solve propositional satisfiability problems, known as SAT-solvers, generally use logical inference techniques, such as unit propagation and clause-learning, to prune the search space.

One of the most surprising empirical findings of the last few years has been the remarkably good performance of general SAT-solvers in solving constraint satisfaction problems. To apply such tools to a constraint satisfaction problem one first has to translate the instance into a set of clauses using some form of Boolean encoding [26,27]. Such encoding techniques tend to obscure the structure of the original problem, and may introduce a very large number of Boolean variables and clauses to encode quite easily-stated constraints. Nevertheless, in quite a few cases, such approaches have out-performed more traditional constraint solving tools [4,3,24].

In this paper we draw on a number of recent analytical approaches to try to account for the good performance of general SAT-solvers on many forms of

constraint problems. Building on the results of [6,7,18], we show that the power of using k -consistency techniques in a constraint problem is precisely captured by using a single inference rule in a standard Boolean encoding of that problem. We refer to this inference rule as *positive-hyper-resolution*, and show that any conclusions deduced by enforcing k -consistency can be deduced by a sequence of positive-hyper-resolution inferences involving Boolean clauses in the original instance and positive-hyper-resolvents with at most k literals. Furthermore, by using the approach of [5] and [25], we show that current clause-learning SAT-solvers will make all such deductions in polynomial expected time, even with a random branching strategy. Hence we show that, although they are not explicitly designed to do so, running a clause-learning SAT-solver on the simplest encoding of a constraint problem efficiently simulates the effects of enforcing k -consistency for *all* values of k .

2 Preliminaries

Definition 2.1. *An instance of the Constraint Satisfaction Problem (CSP) is specified by a triple (V, D, C) , where*

- V is a finite set of variables;
- $D = \{D_v \mid v \in V\}$ where each set D_v is the set of possible values for the variable v , called the domain of v ;
- C is a finite set of constraints. Each constraint in C is a pair (R_i, S_i) where
 - S_i is an ordered list of m_i variables, called the constraint scope;
 - R_i is a relation over D_v of arity m_i , called the constraint relation.

Given any CSP instance (V, D, C) , a *partial assignment* is a mapping f from some subset W of V to $\bigcup D$ such that $f(v) \in D_v$ for all $v \in W$. A partial assignment *satisfies the constraints* of the instance if, for all $(R, (v_1, v_2, \dots, v_m)) \in C$ such that $v_j \in W$ for $j = 1, 2, \dots, m$, we have $(f(v_1), f(v_2), \dots, f(v_m)) \in R$. A partial assignment that satisfies the constraints of an instance is called a *partial solution*¹ to that instance. The set of variables on which a partial assignment f is defined is called the domain of f , and denoted $Dom(f)$. A partial solution f' *extends* a partial solution f if $Dom(f') \supseteq Dom(f)$ and $f'(v) = f(v)$ for all $v \in Dom(f)$. A partial solution with domain V is called a solution.

One way to derive new information about a CSP instance, which may help to determine whether or not it has a solution, is to use some form of constraint propagation to enforce some level of *local consistency* [11]. For example, it is possible to use the notion of *k-consistency*, as in the next definition. We note that there are several different but equivalent ways to define and enforce k -consistency described in the literature [11,13,16]. Our presentation follows [6], which is inspired by the notion of existential k -pebble games introduced by Kolaitis and Vardi [20].

¹ Note that not all partial solutions extend to solutions.

Definition 2.2. [6] For any CSP instance P , the k -consistency closure of P is the set H of partial assignments which is obtained by the following algorithm:

1. Let H be the collection of all partial solutions f of P with $|Dom(f)| \leq k + 1$;
2. For every $f \in H$ with $|Dom(f)| \leq k$ and every variable v of P , if there is no $g \in H$ such that g extends f and $v \in Dom(g)$, then remove f and all its extensions from H ;
3. Repeat step 2 until H is unchanged.

Note that computing the k -consistency closure according to this definition corresponds precisely to enforcing *strong $k + 1$ -consistency* according to the definitions in [11,13,16].

Throughout this paper, we shall assume that the domain of possible values for each variable in a CSP instance is finite. It is straightforward to show that for any fixed k , and fixed maximum domain size, the k -consistency closure of an instance P can be computed in polynomial time [6,13].

Note that any solution to P must extend some element of the k -consistency closure of P . Hence, if the k -consistency closure of P is empty, for some k , then P has no solutions. The converse is not true in general, but it holds for certain special cases, such as the class of instances whose structure has tree-width bounded by k [6], or the class of instances whose constraints are “0/1/all relations”, as defined in [14], or “connected row-convex” relations, as defined in [15]. For these special kinds of instances it is possible to determine in polynomial time whether or not a solution exists simply by computing the k -consistency closure, for an appropriate choice of k . Moreover, if a solution exists, then it can be constructed in polynomial time by selecting each variable in turn, assigning each possible value, re-computing the k -consistency closure, and retaining an assignment that gives a non-empty result.

The following result gives a useful condition for determining whether the k -consistency closure of a CSP instance is empty.

Lemma 2.3. [20] The k -consistency closure of a CSP instance P is non-empty if and only if there exists a non-empty family H of partial solutions to P such that:

1. If $f \in H$, then $|Dom(f)| \leq k + 1$;
2. If $f \in H$ and f extends g , then $g \in H$;
3. If $f \in H$, $|Dom(f)| \leq k$, and $v \notin Dom(f)$ is a variable of P , then there is some $g \in H$ such that g extends f and $v \in Dom(g)$.

A set of partial solutions H satisfying the conditions described in Lemma 2.3 is sometimes called a *strategy* for the instance P [9,20].

One possible approach to solving a CSP instance is to encode it as a propositional formula over a suitable set of Boolean variables, and then use a program to decide the satisfiability of that formula. Many such programs, known as SAT-solvers, are now available and can often efficiently handle problems with thousands, or sometimes even millions, of Boolean variables [29].

Several different ways of encoding a CSP instance as a propositional formula have been proposed [26,27]. Here we consider only a very straightforward encoding, known as the *direct encoding*. In this encoding, for a CSP instance $P = (V, D, C)$ we introduce a set of Boolean variables of the form x_{vi} for each $v \in V$ and each $i \in D_v$. The Boolean variable x_{vi} will be assigned *True* if and only if the original variable v is assigned the value i . To ensure that at least one value is assigned to each variable v , we include the clause $\bigvee_{i \in D_v} x_{vi}$. To ensure that at most one value is assigned to each variable v , we include all binary clauses of the form $\neg x_{vi} \vee \neg x_{vj}$ for all $i, j \in D_v$ with $i \neq j$. Finally, to ensure that each constraint $(R, S) \in C$ is satisfied, we include a clause $\bigvee_{v \in S} \neg x_{vf(v)}$ for each partial assignment f that does *not* satisfy the constraint.

Given any set of clauses we can often deduce further clauses by applying certain *inference rules*. For example, if we have two clauses of the form $C_1 \vee x$ and $C_2 \vee \neg x$, for some (possibly empty) clauses C_1, C_2 , and some variable x , then we can deduce the clause $C_1 \vee C_2$. This form of inference is known as *propositional resolution*; the resultant clause is called the *resolvent* [12].

In the next section, we shall establish a close connection between the k -consistency algorithm and a form of inference called positive-hyper-resolution, which we define as follows:

Definition 2.4. *If we have a collection of clauses of the form $C_i \vee \neg x_i$ for $i = 1, 2, \dots, r$, where each x_i is a Boolean variable, and a purely positive clause $x_1 \vee x_2 \vee \dots \vee x_r$, then we can deduce the clause $C_1 \vee C_2 \vee \dots \vee C_r$.*

*We call this form of inference **positive-hyper-resolution** and the resultant clause $C_1 \vee C_2 \vee \dots \vee C_r$ the positive-hyper-resolvent.*

Note that positive-hyper-resolution is equivalent to a sequence of standard resolution steps. The reason for introducing positive-hyper-resolution is that it allows us to deduce the clauses we need in a single step without needing to introduce intermediate clauses (which may be longer than the positive-hyper-resolvent). By restricting the size of the clauses we use in this way we are able to obtain better performance bounds for the SAT-solvers.

A positive-hyper-resolution *derivation* of a clause C from a set of initial clauses Φ is a sequence of clauses C_1, C_2, \dots, C_m , where $C_m = C$ and each C_i follows by the positive-hyper-resolution rule from some collection of clauses, each of which is either contained in Φ or else occurs earlier in the sequence. The *width* of this derivation is defined to be the maximum size of any of the clauses C_i . If C_m is the empty clause, then we say that the derivation is a *positive-hyper-resolution refutation* of Φ .

3 k -Consistency and Positive-Hyper-Resolution

It has been pointed out by many authors that enforcing local consistency is a form of inference on relations analogous to the use of the resolution rule on clauses [8,11,12,18,19]. On the direct encoding of a CSP instance, our positive-hyper-resolution rule corresponds to the “nogood resolution” rule defined in [18].

The precise strength of the standard resolution inference rule on the direct encoding of a CSP instance was considered in [26], where it was shown that *unit* resolution (where one of the clauses being resolved consists of a single literal), corresponds to enforcing a weak form of local consistency known as *forward checking*. In [18] it was pointed out that the standard resolution rule with no restriction on clause length is able to simulate all the inferences made by a k -consistency algorithm. In [7] it was shown that the standard resolution rule restricted to clauses with at most k literals can be characterised in terms of the Boolean existential $(k + 1)$ -pebble game. It follows that on CSP instances with Boolean domains this form of inference corresponds to enforcing k -consistency.

Here we extend these results a little, to show that for CSP instances with arbitrary finite domains, applying the positive-hyper-resolution rule on the direct encoding to obtain clauses with at most k literals corresponds precisely to enforcing k -consistency. Note that the bound, k , that we impose on the size of the positive-hyper-resolvents, is independent of the domain size. In other words, using this inference rule we only need to consider inferred clauses of size at most k , even though we make use of clauses in the encoding whose size is equal to the domain size, which may be arbitrarily large.

Theorem 3.1. *The k -consistency closure of a CSP instance P is empty if and only if its direct encoding as a set of clauses has a positive-hyper-resolution refutation of width at most k .*

The proof is broken down into two lemmas inspired by Lemmas 2 and 3 in [7].

Lemma 3.2. *Let P be a CSP instance, and let Φ be its direct encoding as a set of clauses. If Φ has no positive-hyper-resolution refutation of width k or less, then the k -consistency closure of P is non-empty.*

Proof. Let V be the set of variables of P , where each $v \in V$ has domain D_v , and let $X = \{x_{vi} \mid v \in V, i \in D_v\}$ be the corresponding set of Boolean variables in Φ . Note that the clauses in Φ are either of the form $\bigvee_{i \in D_v} x_{vi}$ for some $v \in V$, or else consist entirely of negative literals. Let Γ be the set of all clauses having a positive-hyper-resolution derivation from Φ of width at most k . By the definition of positive-hyper-resolution and the observation about Φ , every clause in Γ consists entirely of negative literals.

Now let H be the set of all partial assignments for P with domain size at most $k + 1$ that do not falsify any clause in $\Phi \cup \Gamma$ (under the direct encoding).

Consider any element $f \in H$. By the definition of H , f does not falsify any clause of Φ , so by the definition of the direct encoding, every element of H is a partial solution to P . Furthermore, if f extends g , then g is also an element of H , because g makes fewer assignments than f and hence cannot falsify any additional clauses to f .

If Φ has no positive-hyper-resolution refutation of width at most k , then Γ does not contain the empty clause, so H contains (at least) the partial solution with empty domain, and hence H is not empty.

Now let f be any element of H with $|Dom(f)| \leq k$ and let v be any variable of P that is not in $Dom(f)$. For any partial assignment g that extends f and

has $Dom(g) = Dom(f) \cup \{v\}$ we have that either $g \in H$ or else there exists a clause in $\Phi \cup \Gamma$ that is falsified by g . Since g is a partial assignment, any clause C in $\Phi \cup \Gamma$ that is falsified by g , must consist entirely of negative literals. Hence the literals of C must either be of the form $\neg x_{wf(w)}$ for some $w \in Dom(f)$, or else $\neg x_{vg(v)}$. Moreover, any such clause must contain the literal $\neg x_{vg(v)}$, or else it would already be falsified by f .

Assume, for contradiction, that H does not contain any assignment g that extends f and has $Dom(g) = Dom(f) \cup \{v\}$. In that case, we have that, for each $i \in D_v$, $\Phi \cup \Gamma$ contains a clause C_i consisting of negative literals of the form $\neg x_{wf(w)}$ for some $w \in Dom(f)$, together with the literal $\neg x_{vi}$. Now consider the clause, C , which is the positive-hyper-resolvent of these clauses C_i and the clause $\bigvee_{i \in D_v} x_{vi}$. The clause C consists entirely of negative literals of the form $\neg x_{wf(w)}$ for some $w \in Dom(f)$, so it has width at most $|Dom(f)| \leq k$, and hence is an element of Γ . However C is falsified by f , which contradicts the choice of f . Hence we have shown that for all $f \in H$ with $|Dom(f)| \leq k$, and for all $v \in V$, there is some $g \in H$ such that g extends f and $v \in Dom(g)$.

We have shown that H satisfies all the conditions required by Lemma 2.3, so we conclude that the k -consistency closure of P is non-empty. \square

Lemma 3.3. *Let P be a CSP instance, and let Φ be its direct encoding as a set of clauses. If the k -consistency closure of P is non-empty, then Φ has no positive-hyper-resolution refutation of width k or less.*

Proof. Let V be the set of variables of P , where each $v \in V$ has domain D_v , and let $X = \{x_{vi} \mid v \in V, i \in D_v\}$ be the corresponding set of Boolean variables in Φ .

By Lemma 2.3, if the k -consistency closure of P is non-empty, then there exists a non-empty set H of partial solutions to P which satisfies the three properties described in Lemma 2.3.

Now consider any positive-hyper-resolution derivation Γ from Φ of width at most k . We show by induction on the length of this derivation that the elements of H do not falsify any clause in the derivation. First we note that the elements of H are partial solutions, so they satisfy all the constraints of P , and hence do not falsify any clause of Φ . This establishes the base case. Assume, for induction, that all clauses in the derivation earlier than some clause C are not falsified by any element of H .

Since the clauses in Φ are either of the form $\bigvee_{i \in D_v} x_{vi}$ for some $v \in V$, or else consist entirely of negative literals, it follows that any clause in the derivation obtained by positive-hyper-resolution consists entirely of negative literals.

If $f \in H$ falsifies $C \in \Gamma$, then the literals of C must all be of the form $\neg x_{vf(v)}$, for some $v \in Dom(f)$. Hence we may assume, without loss of generality, that C is the positive-hyper-resolvent of a set of clauses $\Delta = \{C_i \vee \neg x_{vi} \mid i \in D_v\}$ and the clause $\bigvee_{i \in D_v} x_{vi}$. Since the width of the derivation is at most k , C contains at most k literals, and hence we may assume that $|Dom(f)| \leq k$. But then, by the choice of H , there must exist some extension g of f in H such that $v \in Dom(g)$. Any such g will falsify some clause in Δ , which contradicts our inductive hypothesis. Hence no $f \in H$ falsifies C , so C cannot be empty.

It follows that no positive-hyper-resolution derivation of width at most k can contain the empty clause. \square

4 Positive-Hyper-Resolution and SAT-Solvers

In this section we adapt the machinery of [5] and [25] to show that for any fixed k , the existence of a positive-hyper-resolution refutation of width k is likely to be discovered by a SAT-solver in polynomial-time using standard clause learning and restart techniques, even with a totally random branching strategy.

Note that previous results about the power of clause-learning SAT-solvers have generally assumed an optimal branching strategy [10,25] - they have shown what solvers are potentially capable of doing, rather than what they are likely to achieve in practice. The exception is [5], which gives an analysis of likely behaviour, but relies on the existence of a standard resolution proof of bounded width. Here we show that the results of [5] can be extended to hyper-resolution proofs, which can be much shorter and narrower than their associated standard resolution proofs.

We will make use of the following terminology from [5]. For a clause C , a Boolean variable x , and a truth value $a \in \{0, 1\}$, the restriction of C by the assignment $x = a$, denoted $C|_{x=a}$, is defined to be the constant $\mathbf{1}$, if the assignment satisfies the clause, or else the clause obtained by deleting from C any literals involving the variable x . For any sequence of assignments S of the form $(x_1 = a_1, x_2 = a_2, \dots, x_r = a_r)$ we write $C|_S$ to denote the result of computing the restriction of C by each assignment in turn. If $C|_S$ is empty, then we say that the assignments in S *falsify* the clause C . For a set of clauses Δ , we write $\Delta|_S$ to denote the set $\{C|_S \mid C \in \Delta\} \setminus \{\mathbf{1}\}$.

Most current SAT-solvers operate in the following way [5,25]. They maintain a database of clauses Δ and a current state S , which is a partial assignment of truth values to the Boolean variables in the clauses of Δ . A high-level description of the algorithms used to update the clause database and the state, derived from the description given in [5], is shown in Algorithm 1 (a similar framework, using slightly different terminology, is given in [25]).

Now consider a run of the algorithm shown in Algorithm 1, started with the initial database Δ , and the empty state S_0 , until it either halts or discovers a *conflict* (i.e., $\emptyset \in \Delta|_S$). Such a run is called a *round started with Δ* , and we represent it by the sequence of states S_0, \dots, S_m , that the algorithm maintains. Note that each state S_i extends the state S_{i-1} by a single assignment to a Boolean variable, which may be either a *decision assignment* or an *implied assignment*.

An initial segment S_0, S_1, \dots, S_r of a round started with Δ is called an *inconclusive partial round* if $\Delta|_{S_r}$ is non-empty, does not contain the empty clause, and does not contain any unit clauses. Note that for any clause $C \in \Delta$, if S_0, S_1, \dots, S_r is an inconclusive partial round started with Δ , and S_r falsifies all the literals of C except one, then it must satisfy the remaining literal, and hence satisfy C . This property of clauses is captured by the following definition.

Algorithm 1. Framework for typical clause-learning SAT-solver

Input: Δ : set of clauses; S : partial assignment of truth values to variables.

```

1. while  $\Delta|_S \neq \emptyset$  do
2.   if  $\emptyset \in \Delta|_S$  then CONFLICT
3.     if  $S$  contains no decision assignments then
4.       print “UNSATISFIABLE” and halt
5.     else
6.       apply the learning scheme to add a new clause to  $\Delta$ 
7.       if restart policy says restart then
8.         set  $S = \emptyset$ 
9.       else
10.        select most recent conflict-causing unreversed decision assignment in  $S$ 
11.        reverse this decision, and remove all later assignments from  $S$ 
12.      end if
13.    end if
14.    else if  $\{l\} \in \Delta|_S$  for some literal  $l$  then UNIT PROPAGATION
15.      add to  $S$  the implied assignment  $x = a$  which satisfies  $l$ 
16.    else DECISION
17.      apply the branching strategy to choose a decision assignment  $x = a$ 
18.      add this decision assignment to  $S$ 
19.    end if
20.  end while
21. print “SATISFIABLE” and output  $S$ 

```

Definition 4.1. [5] Let Δ be a set of clauses, C a non-empty clause, and l a literal of C . We say that Δ absorbs C at l if every inconclusive partial round started with Δ that falsifies $C \setminus \{l\}$ satisfies C .

If Δ absorbs C at each literal l in C , then we simply say that Δ absorbs C .

Note that a clause that is *not* absorbed by a set of clauses Δ is referred to in [25] as *1-empowering* with respect to Δ .

Lemma 4.2. [5] Let Δ and Δ' be sets of clauses, and let C and C' be non-empty clauses.

1. If C belongs to Δ , then Δ absorbs C ;
2. If $C \subseteq C'$ and Δ absorbs C , then Δ absorbs C' ;
3. If $\Delta \subseteq \Delta'$ and Δ absorbs C , then Δ' absorbs C ;
4. If $\Delta \subseteq \Delta''$ and Δ absorbs C and Δ entails Δ'' , then Δ'' absorbs C .

To allow further analysis, we need to make some assumptions about the *learning scheme*, the *restart policy* and the *branching strategy* used by our SAT-solver.

The learning scheme is a rule that creates and adds a new clause to the database whenever there is a conflict. Such a clause is called a *conflict clause*, and each of its literals is falsified by some assignment in the current state. If a literal is falsified by the i -th decision assignment, or some later implied assignment before

$(i+1)$ -th decision assignment, it is said to be falsified at level i . If a conflict clause contains exactly one literal that is falsified at the maximum possible level, it is called an *asserting clause* [28,25].

Assumption 1. *The learning scheme chooses an asserting clause.*

Most learning schemes in current use satisfy this assumption [28,25], including the learning schemes called “1UIP” and “DECISION” described in [28].

We make no particular assumption about the restart policy. However, our main result is phrased in terms of a bound on the expected number of restarts. If the algorithm restarts after r conflicts, our bound on the expected number of restarts can simply be multiplied by r to get a bound on the expected number of conflicts. This means that the implications will be strongest if the algorithm restarts *immediately after each conflict*. In that case, $r = 1$ and our bound will also bound the expected number of conflicts. Existing SAT-solvers typically do not employ such an aggressive restart policy, but we note the remark in [25] that “there has been a clear trend towards more and more frequent restarts for modern SAT solvers”.

The branching strategy determines which decision assignment is chosen after an inconclusive partial round. In most current SAT solvers the strategy is based on some heuristic measure of *variable activity*, which is related to the occurrence of a variable in a conflict clause [23]. However, to simplify the probabilistic analysis, we will make the following assumption.

Assumption 2. *The branching strategy chooses a variable uniformly at random amongst the unassigned variables, and assigns it the value TRUE.*

As noted in [5], the same analysis we give below can also be applied to any other branching strategy that randomly chooses between making a heuristic-based decision or a randomly-based decision, provided that the second case has non-negligible probability p . In that case, the bounds we obtain on the expected number of restarts can simply be multiplied by p^{-k} .

An algorithm that behaves according to the description in Algorithm 1, and satisfies the assumptions above, will be called a *standard randomised SAT-solver*.

Theorem 4.3. *If a set of non-empty clauses Δ over n Boolean variables has a positive-hyper-resolution refutation of width k and length m , where all derived clauses contain only negative literals, then the expected number of restarts required by a standard randomised SAT-solver to discover that Δ is unsatisfiable is less than $mnk^2 \binom{n}{k}$.*

Proof. Let C_1, C_2, \dots, C_m be a positive-hyper-resolution refutation of width k from Δ , where each C_i contains only negative literals, and C_m is the first occurrence of the empty clause. Since each clause in Δ is non-empty, C_m must be derived by positive-hyper-resolution from some collection of negative literals $\neg x_1, \neg x_2, \dots, \neg x_d$ and a purely positive clause $x_1 \vee x_2 \vee \dots \vee x_d$.

Now consider a standard SAT-solver started with database Δ . Once all of the unit clauses $\neg x_i$ are absorbed by the current database, then, by Definition 4.1,

any further inconclusive partial round of the algorithm must assign all variables x_i false, and hence falsify the clause $x_1 \vee x_2 \vee \dots \vee x_d$. Since this happens even when no decision assignments are made, the SAT-solver will report unsatisfiability.

It only remains to bound the expected number of restarts required until each clause C_i is absorbed, for $1 \leq i < m$. Let each C_i be the positive-hyper-resolvent of clauses $C_{i1}, C_{i2}, \dots, C_{id}$, each of the form $C'_{ij} \vee \neg x_j$, together with a purely positive clause $C_{i0} = x_1 \vee x_2 \vee \dots \vee x_d$ from Δ . Assume also that each clause C'_{ij} is absorbed by Δ .

If Δ absorbs C_i , then no further learning or restarts are needed, so assume now that Δ does not absorb C_i . By Definition 4.1, this means that there exists some literal l and some inconclusive partial round R started with Δ , that falsifies $C_i \setminus \{l\}$ and does not satisfy C_i . Note that R must leave the literal l unassigned, because one assignment would satisfy C_i and the other would force all of the literals $\neg x_j$ used in the positive-hyper-resolution step to be satisfied, because each C'_{ij} is absorbed by Δ , so C_{i0} would be falsified, contradicting the fact that R is inconclusive.

Hence, if the branching strategy chooses to falsify the literals $C_i \setminus \{l\}$ whenever it has a choice, it will construct an inconclusive partial round R' where l is unassigned (since all the decision assignments in R' are also assigned the same values in R , any implied assignments in R' must also be assigned the same values² in R , but we have shown that R leaves l unassigned). If the branching strategy then chooses to falsify the remaining literal l of C_i , then the algorithm would construct a complete round R'' where C_{i0} is falsified, and all decision assignments falsify literals in C_i . Hence, by Assumption 1, the algorithm would then learn some asserting clause C' and add it to Δ to obtain a new set Δ' .

Since C' is an asserting clause, it contains exactly one literal, l' , that is falsified at the highest level in R'' . Hence, any inconclusive partial round R started with Δ' that falsifies $C_i \setminus \{l\}$ will falsify all but one literal of C' , and hence force the remaining literal l' to be satisfied, by unit propagation. If this new implied assignment for l' propagates to force l to be true, then R satisfies C_i , and hence Δ' absorbs C_i at l . If not, then the branching strategy can once again choose to falsify the remaining literal l of C_i , which will cause a new asserting clause to be learnt and added to Δ . Since each new asserting clause forces a new literal to be satisfied after falsifying $C_i \setminus \{l\}$ this process can be repeated fewer than n times before it is certain that Δ' absorbs C_i at l .

Now consider any sequence of k random branching choices. If the first $k - 1$ of these each falsify a literal of $C_i \setminus \{l\}$, and the final choice falsifies l , then we have shown that the associated round will reach a conflict, and add an asserting clause to Δ . With a random branching strategy, as described in Assumption 2, the probability that this happens is at least the probability that the first $k - 1$ random choices consist of a fixed set of variables (in some order), and the final choice is the variable associated with l . The number of random choices that fall in a fixed set follows the hypergeometric distribution, so the overall probability of this is $\frac{1}{\binom{n}{k-1}} \frac{1}{(n-k+1)} = 1/(k \binom{n}{k})$.

² See Lemma 3 of [5] for a more formal statement and proof.

To obtain an upper bound on the expected number of restarts, consider the worst case where we require n asserting clauses to be added to absorb each clause C_i at each of its k literals l . Since we require only an upper bound, we will treat each round as an independent trial with success probability $p = 1/(k\binom{n}{k})$, and consider the worst case where we have to achieve $(m-1)nk$ separate consecutive successes to ensure that C_i for $1 \leq i < m$ is absorbed. In this case the total number of restarts will follow a negative binomial distribution, with expected value $(m-1)nk/p$. Hence in all cases the expected number of restarts is less than $mnk^2\binom{n}{k}$. \square

A tighter bound on the number of restarts can be obtained if we focus on the DECISION learning scheme [5,28], as the next result indicates.

Theorem 4.4. *If a set of non-empty clauses Δ over n Boolean variables has a positive-hyper-resolution refutation of width k and length m , where all derived clauses contain only negative literals, then the expected number of restarts required by a standard randomised SAT-solver using the DECISION learning scheme to discover that Δ is unsatisfiable is less than $m\binom{n}{k}$.*

Proof. The proof is similar to the proof of Theorem 4.3, except that the DECISION learning scheme has the additional feature that the literals in the chosen conflict clause falsify a subset of the current decision assignments. Hence in the situation we consider, where the decision assignments all falsify literals of some clause C_i (in any order), this learning scheme will learn a subset of C_i , and hence immediately absorb C_i , by Lemma 4.2(1,2). Hence the maximum number of learnt clauses required is reduced from $(m-1)nk$ to $(m-1)$, and the probability is increased from $1/(k\binom{n}{k})$ to $1/\binom{n}{k}$, giving the tighter bound. \square

Note that a similar argument shows that the standard deviation of the number of restarts is less than the standard deviation of a negative binomial distribution with parameters m and $1/\binom{n}{k}$, which is less than $\sqrt{m\binom{n}{k}}$. Hence, by Chebyshev's inequality (one-tailed version), the probability that a standard randomised SAT-solver using the decision learning scheme will discover that Δ is unsatisfiable after $(m + \sqrt{m})\binom{n}{k}$ restarts is greater than $1/2$.

5 k -Consistency and SAT-Solvers

By combining Theorem 3.1 and Theorem 4.4 we obtain the following result linking k -consistency and SAT-solvers.

Theorem 5.1. *If the k -consistency closure of a CSP instance P is empty, then the expected number of restarts required by a standard randomised SAT-solver using the DECISION learning scheme to discover that the direct encoding of P is unsatisfiable is $O(n^{2k}d^{2k})$, where n is the number of variables in P and d is the maximum domain size.*

Proof. The length m of a positive-hyper-resolution refutation of width k is bounded by the number of possible no-goods of length at most k for P , which is $\sum_{i=1}^k d^i \binom{n}{i}$. Hence, by Theorem 3.1 and Theorem 4.4 we obtain a bound of $\left(\sum_{i=1}^k d^i \binom{n}{i}\right) \binom{nd}{k}$, which is $O(n^{2k} d^{2k})$. \square

Hence a standard randomised SAT-solver with a suitable learning strategy will decide the satisfiability of any CSP instance with tree-width k with $O(n^{2k} d^{2k})$ expected restarts, even when it is set to restart immediately after each conflict. In particular, the satisfiability of any tree-structured CSP instance (i.e., with tree-width 1) will be decided by such a solver with at most $O(n^2 d^2)$ expected conflicts, which is comparable with the growth rate of an optimal arc-consistency algorithm. Note that this result cannot be obtained directly from [5], because the direct encoding of an instance with tree-width k is a set of clauses whose tree-width may be as high as dk .

Moreover, a standard randomised SAT-solver will decide the satisfiability of any CSP instance, with any structure, within the same polynomial bounds, if the constraint relations satisfy certain algebraic properties that ensure bounded width [9]. Examples of such constraint types include the “0/1/all relations”, defined in [14], and the “connected row-convex” relations, defined in [15], which can both be decided by 2-consistency.

6 Experimental Results

The bounds we obtain in this paper are very conservative, and are likely to be met very easily in practice.

To investigate how an existing SAT-solver performs in practice, we measured the performance of the MiniSAT solver [2] version 2-070721 on a family of CSP instances that can be decided by a fixed level of consistency. We ran the experiments with preprocessing switched off, in order to get a solver that uses only unit propagation and conflict-directed learning with restarts.

We also modified the MiniSAT solver to follow the random branching strategy described above. Our modified solver does not delete any learnt clauses and uses an extreme restart policy that makes it restart whenever it encounters a conflict. We refer to this modified solver as simple-MiniSAT.

For all of the results, the times given are elapsed times on a Lenovo 3000 N200 laptop with an Intel Core 2 Duo processor running at 1.66GHz with 2GB of RAM. For our simple-MiniSAT solver, each generated instance was run three times and the mean times and mean number of restarts are shown.

Example 6.1. We consider a family of instances specified by two parameters, w and d . They have $((d-1) * w + 2) * w$ variables arranged in groups of size w , each with domain $\{0, \dots, d-1\}$. We impose a constraint of arity $2w$ on each pair of successive groups, requiring that the sum of the values assigned to the first of these two groups should be strictly smaller than the sum of the values assigned to the second. This ensures that the instances generated are unsatisfiable. An

instance with $w = 2$ and $d = 2$ is shown diagrammatically and defined using the specification language MiniZinc [1] in Figure 1.

The structure of these instances has a simple tree-decomposition as a path of nodes, with each node corresponding to a constraint scope. Hence the tree-width of these instances is $2w - 1$, and they can be shown to be unsatisfiable by enforcing $2w - 1$ consistency. However, these instances cannot be solved efficiently using standard propagation algorithms which only prune individual domain values.

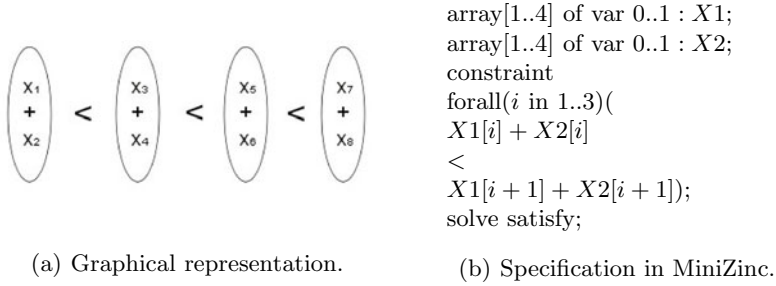


Fig. 1. An example of a CSP instance with $w = 2$, $d = 2$ and tree-width = 3

Table 1. Performance of CP-solvers and SAT-solvers on instances from Example 6.1

group size (w)	domain size (d)	CSP variables (n)	Minion (sec)	G12 (sec)	MiniSAT (sec)	simple-MiniSAT (sec)	simple-MiniSAT restarts
2	2	8	0.010	0.238	0.004	0.004	17
2	3	12	0.012	0.246	0.007	0.008	175
2	4	16	0.026	0.273	0.021	0.038	866
2	5	20	0.043	0.525	0.052	0.146	2 877
2	6	24	1.040	6.153	0.157	0.626	7 582
2	7	28	47.554	205.425	0.433	2.447	17 689
2	8	32	> 20 min	> 20 min	1.273	10.169	35 498
2	9	36	> 20 min	> 20 min	3.301	44.260	65 598
2	10	40	> 20 min	> 20 min	8.506	135.215	108 053
3	2	15	0.012	0.240	0.005	0.008	176
3	3	24	0.370	1.120	0.103	0.377	4 839
3	4	33	> 20 min	> 20 min	1.942	22.357	43 033
3	5	42	> 20 min	> 20 min	29.745	945.202	209 094

Table 1 shows the runtimes of simple-MiniSAT and the original MiniSAT solver on this family of instances, along with times for two state-of-the-art CP solvers: Minion [17] and G12 [1]. Note that MiniSAT is remarkably effective in solving these instances, compared to the CP solvers, even though they are encoded into a large number of clauses with a much larger tree-width than the original instance. Although our modified SAT solver takes a little longer, it still performs better on

these instances than the CP solvers and the number of restarts (and hence the number of conflicts) is much lower than the polynomial upper bound obtained in Theorem 5.1 (see Figure 2).

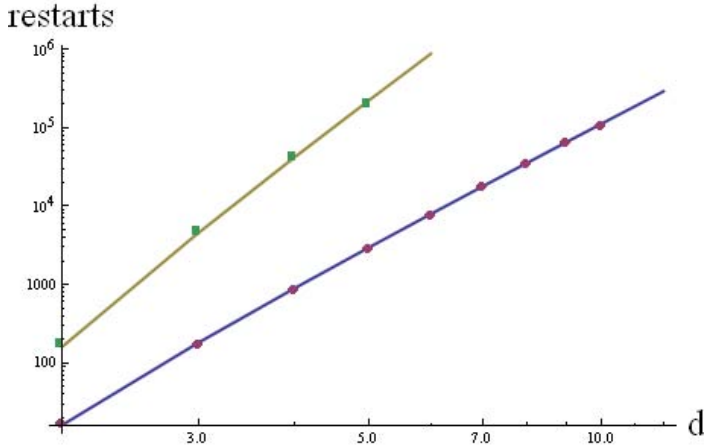


Fig. 2. Log-log plot of the number of restarts/conflicts used by simple-MiniSAT on the instances from Example 6.1. Circles show values for $w = 2$; squares show values for $w = 3$; solid lines show the functions $d^2 \binom{n/2}{3}$ (lower line) and $d^4 \binom{n/3}{3}$ (upper line). Note that these experimentally determined growth functions are much lower than the worst-case bound calculated in Theorem 5.1.

7 Conclusion

We have shown that the notion of k -consistency can be precisely captured by a single inference rule on the direct encoding of a CSP instance, restricted to deriving only clauses with at most k literals. We used this to show that a clause-learning SAT-solver with a purely random branching strategy will simulate the effect of enforcing k -consistency in expected polynomial time, for all fixed k . This is sufficient to ensure that such solvers are able to solve certain problem families much more efficiently than conventional CP solvers relying on GAC-propagation.

In principle clause-learning SAT-solvers can also do much more. It is known that, with an appropriate branching strategy and restart policy, they are able to p-simulate general resolution [10,25], and general resolution proofs can be exponentially shorter than the negative resolution proofs we have considered here [18]. In practice, it seems that current clause-learning SAT-solvers with highly-tuned learning schemes, branching strategies and restart policies are often able to exploit structure in the encoding of a CSP instance even more effectively than local consistency techniques. Hence considerable work remains to be done in understanding the relevant features of instances which they are able to exploit, in order to predict their effectiveness in solving different kinds of CSP instances.

References

1. G12/MiniZinc constraint solver. Software, <http://www.g12.cs.mu.oz.au/minizinc/download.html>
2. MiniSat solver. Software, <http://minisat.se/MiniSat.html>
3. 2nd internat. CSP solver competition, <http://www.cril.univ-artois.fr/CPAI06/>
4. 3rd international CSP solver competition, <http://cpai.ucc.ie/08/>
5. Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 114–127. Springer, Heidelberg (2009)
6. Atserias, A., Bulatov, A.A., Dalmau, V.: On the power of k -consistency. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 279–290. Springer, Heidelberg (2007)
7. Atserias, A., Dalmau, V.: A combinatorial characterization of resolution width. *Journal of Computer and Systems Science* 74(3), 323–334 (2008)
8. Bacchus, F.: GAC Via Unit Propagation. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 133–147. Springer, Heidelberg (2007)
9. Barto, L., Kozik, M.: Constraint satisfaction problems of bounded width. In: Proceedings of FOCS 2009, pp. 595–603. IEEE Computer Society, Los Alamitos (2009)
10. Beame, P., Kautz, H.A., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research* 22, 319–351 (2004)
11. Bessière, C.: Constraint propagation. In: *Handbook of Constraint Programming*, ch. 3, pp. 29–83. Elsevier, Amsterdam (2006)
12. Bordeaux, L., Hamadi, Y., Zhang, L.: Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys* 38(4) (2006)
13. Cooper, M.C.: An optimal k -consistency algorithm. *Artificial Intelligence* 41, 89–95 (1989)
14. Cooper, M.C., Cohen, D.A., Jeavons, P.G.: Characterising tractable constraints. *Artificial Intelligence* 65, 347–361 (1994)
15. Deville, Y., Barette, O., van Hentenryck, P.: Constraint satisfaction over connected row convex constraints. In: Proceedings of IJCAI 1997, pp. 405–411 (1997)
16. Freuder, E.C.: Synthesizing constraint expressions. *ACM Comm.* 21, 958–966 (1978)
17. Gent, I., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: Proceedings of ECAI 2006, pp. 98–102. IOS Press, Amsterdam (2006)
18. Hwang, J., Mitchell, D.: 2-way vs. d -way branching for CSP. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 343–357. Springer, Heidelberg (2005)
19. Rish, I., Dechter, R.: Resolution versus search: Two strategies for SAT. *Journal of Automated Reasoning* 24(1/2), 225–275 (2000)
20. Kolaitis, P.G., Vardi, M.Y.: A game-theoretic approach to constraint satisfaction. In: Proceedings of AAAI 2000, pp. 175–181 (2000)
21. Mackworth, A.K.: Consistency in networks of relations. *Artificial Intelligence* 8, 99–118 (1977)
22. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences* 7, 95–132 (1974)
23. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: International Design Automation Conference, DAC, pp. 530–535 (2001)

24. Petke, J., Jeavons, P.G.: Tractable benchmarks for constraint programming. Technical Report RR-09-07, Computing Laboratory, University of Oxford (2009)
25. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers with restarts. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 654–668. Springer, Heidelberg (2009)
26. Walsh, T.: SAT v CSP. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 441–456. Springer, Heidelberg (2000)
27. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. *Constraints* 14(2), 254–272 (2009)
28. Zhang, L., Madigan, C.F., Moskewicz, M.W., Andmalik, S.: Efficient conflict driven learning in a Boolean satisfiability solver. In: Proceedings of the International Conference on Computer-Aided Design (ICCAD 2001), pp. 279–285 (2001)
29. Zhang, L., Malik, S.: The quest for efficient Boolean satisfiability solvers. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 641–653. Springer, Heidelberg (2002)

Heuristics for Planning with SAT

Jussi Rintanen

NICTA and the Australian National University
Canberra, Australia

Abstract. Generic SAT solvers have been very successful in solving hard combinatorial problems in various application areas, including AI planning. There is potential for improved performance by making the SAT solving process more application-specific. In this paper we propose a variable selection strategy for AI planning. The strategy is based on generic principles about properties of plans, and its performance with standard planning benchmarks often substantially improves on generic variable selection heuristics used in SAT solving, such as the VSIDS strategy. These improvements lift the efficiency of SAT based planning to the same level as best planners that use other search methods.

1 Introduction

Planning is one of several problems that have been successfully solved with SAT algorithms [1]. Most works have used a generic SAT solver, which recently use the conflict-directed clause learning (CDCL) algorithm and the VSIDS heuristic [2].

In this work we investigate SAT solving for planning with CDCL but with a heuristic that radically differs from VSIDS and is based on a simple property all solutions to a planning problem have to satisfy. The work is motivated by the need to better understand why SAT solvers are successful in solving AI planning and related problems, and by the need and opportunity to develop more powerful, problem-specific heuristics for SAT.

Our heuristic chooses action variables that contribute to the goals or (current) subgoals. The principle is extremely simple: *for a given (sub)goal, choose an action that achieves the (sub)goal and that can be taken at the earliest time in which the (sub)goal can become (and remain) true*. After choosing an action, its preconditions become new subgoals, for which supporting actions are found in the same way. This principle is easy to implement: start from a goal (or a subgoal), go backwards step by step until a time point in which the goal is *false*, and choose any of the actions that achieve the goal and are possible at that time point. If such an action existed in the plan already, perform the procedure recursively with the preconditions of the action as the subgoals.

The above principle is so simple that it is surprising that it alone, without any further heuristics, is, for satisfiable problems representing standard planning benchmark problems, very often far more effective than the VSIDS heuristic found in the best current SAT solvers. Furthermore, the new heuristic lifts the efficiency of SAT-based planning to the same level with the best existing planning algorithms that are based on other search methods, including heuristic state space search. This result is extremely surprising because the currently best state-space search planners, which have their origins in the work of Bonet and Geffner [3] more than ten years ago, use far more complex

heuristics, and were in most cases developed with the very benchmark problems in mind which they are usually evaluated on. In contrast, our heuristic relies on a very natural and simple principle that represents common intuitions about planning.

We view the new heuristic as a step toward developing better SAT-based techniques for planning and other related problems such as model-checking and discrete-event systems diagnosis. More advanced heuristics for these applications are likely to also incorporate features from VSIDS, including the computation of *weights* of variables based on their occurrence in recently learned clauses.

As already mentioned, the new heuristic is better than VSIDS with satisfiable formulas representing standard planning benchmarks, and worse with unsatisfiable ones. Heuristics in SAT solving have two complementary roles. For satisfiable formulas, heuristics help in finding a satisfying assignment quickly, hopefully avoiding much of the brute-force search resulting from wrong choices of variables and truth-values. This is similar to heuristics in state-space search for planning which help in finding a path (a state sequence) to a goal state. For an unsatisfiable formula, heuristics help the solver to efficiently find a *refutation proof*. Our heuristic only indirectly addresses unsatisfiability proofs by avoiding actions that cannot possibly contribute to the refutation proof. Heuristics should more explicitly try to make finding the unsatisfiability proofs easier, but we did not have this as an objective for our new heuristics.

The two roles of heuristics in SAT solving are both important, but their importance depends on the type of planning being done. If the objective is to find an optimal plan, one has to *prove* that no better plans exist. For this one needs efficient detection of unsatisfiability. If the objective is to just find a plan, with no quality guarantees, the importance of efficient unsatisfiability proofs is much smaller. The experiments we will be presenting later address this duality. We leave the problem of improving the efficiency of unsatisfiability for future work.

In addition to improved efficiency of finding satisfying valuations, another benefit our variable selection heuristic has over VSIDS, from the point of view of planning research, is that it is easy to understand, and there are obvious avenues to improving the scheme, and obvious avenues for expressing follow-up questions about it, for example concerning the difficulty of finding unsatisfiability proofs. Addressing these things in the context of the VSIDS heuristics seems quite a bit trickier because of a lack of intuitive explanations of what VSIDS does in terms of search for plans.

The structure of the paper is as follows. Sect. 2 describes the background of the work in planning with SAT. In Sect. 3 we explain the CDCL algorithm. In Sect. 4 we present the variable selection scheme for planning, experimentally evaluate it in Sect. 5 and discuss related work in Sect. 6 before concluding the paper in Sect. 7.

2 Planning as Satisfiability

The classical planning problem involves finding a sequence of actions from a given initial state to a goal state. The actions are deterministic, which means that an action and the current state determine the successor state uniquely. In the simplest formalization of planning actions are pairs (p, e) where p and e are consistent sets of propositional literals on the finite set A of *state variables*, respectively called *the precondition* and *the effects*.

Actions described this way are often known as STRIPS actions for historical reasons. An action (p, e) is *executable* in a state s if $s \models p$. A state $s : A \rightarrow \{0, 1\}$ is a valuation of all state variables. For a given state s and an action (p, e) which is executable in s , the unique successor state $s' = \text{exec}_{(p,e)}(s)$ is determined by $s' \models e$ and $s'(a) = s(a)$ for all $a \in A$ such that a does not occur in e . This means that the effects are true in the successor state and all state variables not affected by the action retain their values. Given an initial state I , a solution to the planning problem with goal G (a set of literals) is a sequence of actions o_1, \dots, o_n such that $\text{exec}_{o_n}(\text{exec}_{o_{n-1}}(\dots \text{exec}_{o_2}(\text{exec}_{o_1}(I)) \dots)) \models G$.

Kautz and Selman [1] proposed finding plans by reduction to SAT. The reduction is similar to the reduction of NP Turing machines to SAT used in Cook's proof of NP-hardness of SAT [4]. The reduction is parameterized by a horizon length $T \geq 0$. The value of each state variable $a \in A$ in each time point $t \in \{0, \dots, T\}$ is represented by a propositional variable $a@t$. Additionally, it is often useful to have the actions represented as propositional variables, so for each action o and $t \in \{0, \dots, T-1\}$ we similarly have a propositional variable $o@t$ indicating whether action o is taken at t .

In this paper, we use an encoding of planning which allows several actions to be taken at the same time point in parallel. The "parallelism" (partial ordering) is allowed when it is possible to totally order the actions to a sequential plan as defined above. There are different possibilities in defining this kind of parallel plans [5,6].

To represent planning as a SAT problem, each action $o = (p, e)$ and time point $t \in \{0, \dots, T-1\}$ is mapped to formulas $o@t \rightarrow \bigwedge_{l \in p} l@t$ and $o@t \rightarrow \bigwedge_{l \in e} l@(t+1)$.¹ These two formulas respectively correspond to the executability condition and the first part of the definition of successor states. The second part, about state variables that do not change, is encoded as follows, for the case in which in several actions can be taken in parallel. For each state variable $a \in A$ we have the formula

$$(\neg a@t \wedge a@(t+1)) \rightarrow (o_1^a@t \vee \dots \vee o_n^a@t)$$

where o_1^a, \dots, o_n^a are all the actions which have a as an effect. Similarly we have for each $a \in A$ the formula

$$(a@t \wedge \neg a@(t+1)) \rightarrow (o_1^{-a}@t \vee \dots \vee o_m^{-a}@t)$$

for explaining possible changes from true to false, where $o_1^{-a}, \dots, o_m^{-a}$ are all the actions with $\neg a$ as an effect. These formulas (often called *the frame axioms*) allow to infer that a state variable does not change if none of the actions changing it is taken.

Finally, to rule out solutions that don't correspond to any plan because parallel actions cannot be serialized,² further formulas are needed. In this work we have used the linear-size \exists -step semantics encoding of Rintanen et al. [5].

There is one more component in efficient SAT encodings of planning, which is logically redundant but usually critical for efficiency: *invariants* (mutexes). Invariants $l \vee l'$ express dependencies between state variables. Many of the standard planning benchmarks represent multi-valued state variables in terms of several Boolean ones, and a

¹ For negative literals $l = \neg a$, $l@t$ means $\neg(a@t)$, and for positive literals $l = a$ it means $a@t$.

² For example, actions $(\{a\}, \{\neg b\})$ and $(\{b\}, \{\neg a\})$ cannot be made to a valid sequential plan, because taking either action first would falsify the precondition of the other.

typical invariant $\neg x_a \vee \neg x_b$ says that a multi-valued variable x can only have one of the values a and b . To compute these invariants, we used the algorithm of Rintanen [7] which is defined for the general (ground) PDDL language which includes STRIPS.

For a given set A of state variables, initial state I , set O of actions, goals G and horizon length T , we can compute (in linear time in the product of T and the sum of sizes of A , I , O and G) a formula Φ_T such that $\Phi_T \in \text{SAT}$ if and only if there is a plan with horizon $0, \dots, T$. Φ_T includes the formulas described above, and for all $a \in A$ the unit clause $a@0$ if $I(a) = 1$ and $\neg a@0$ if $I(a) = 0$, and $l@T$ for all $l \in G$. These formulas are in CNF after trivial rewriting.

A planner can do the tests $\Phi_0 \in \text{SAT}$, $\Phi_1 \in \text{SAT}$, $\Phi_2 \in \text{SAT}$, and so on, sequentially one by one, or it can make several of these tests in parallel (interleave them). For this we will later be using Algorithm B of Rintanen et al. [5] which allocates CPU time to different horizon lengths according to a decreasing geometric series, so that horizon length $t+1$ gets γ times the CPU the horizon length t gets, for some fixed $\gamma \in]0, 1]$. In general, the parallelized strategies can be orders of magnitudes faster than the sequential strategy because they do not need to complete the test $\Phi_t \in \text{SAT}$ (finding Φ_t unsatisfiable) before proceeding with the test $\Phi_{t+1} \in \text{SAT}$. This explains why, in this setting, it is far more important to efficiently determine satisfiability than unsatisfiability.

3 The CDCL Algorithm for SAT

In this section we briefly describe the standard conflict-directed clause learning (CDCL) algorithm for the SAT problem. This algorithm is the basis of most of the currently leading SAT solvers in the zChaff family [2].

For a general overview of the CDCL algorithm see standard references [8,9]. The main loop of the CDCL algorithm (see Fig. 1) chooses an unassigned variable, assigns a truth-value to it, and then performs unit propagation to extend the current valuation v with forced variable assignments that directly follow from the existing valuation by the unit resolution rule. If one of the clauses is falsified, a new clause which would have prevented the current valuation is learned. This new clause is a logical consequence of the original clause set. Then, some of the last assignments are undone, and the assign-infer-learn cycle is repeated. The procedure ends when the empty clause has been learned (no valuation can satisfy the clauses) or a satisfying valuation has been found.

The selection of the decision variable (line 5) and its value (line 6) can be arbitrary (without compromising the correctness of the algorithm), and can therefore be based on a heuristic. The heuristic is critical for the efficiency of the CDCL algorithm. On line 7 the standard unit propagation algorithm is run. It infers a forced assignment for a variable x if there is a clause $x \vee l_1 \vee \dots \vee l_n$ or $\neg x \vee l_1 \vee \dots \vee l_n$ and $v \models \neg l_1 \vee \dots \vee \neg l_n$.

The inference of a new clause on line 10 is the key component of CDCL. The clause will prevent generating the same unsatisfying assignment again, leading to traversing a different part of the search space.

4 A New Heuristic for Planning

The goal of the current work is to present a new way of choosing the decision variables (lines 5 and 6 in the CDCL procedure in Fig. 1) specific to planning. Generic CDCL

```

1: procedure CDCL( $C$ )
2: Initialize  $v$  to satisfy all unit clauses in  $C$ ;
3: extend  $v$  by unit propagation with  $v$  and  $C$ ;
4: while  $C$  does not contain the empty clause do
5:   choose a variable  $a$  with  $v(a)$  unassigned;
6:   assign  $v(a) := 1$  or  $v(a) := 0$ ;
7:   extend  $v$  by unit propagation with  $v$  and  $C$ ;
8:   if  $v$  falsifies a clause in  $C$ 
9:     then
10:    infer a new clause  $c$  and add it to  $C$ ;
11:    undo assignments until  $a$  so that  $c$  is not falsified;
12:   end if
13: end while

```

Fig. 1. Outline of the CDCL algorithm

solvers choose the decision variables based on the variables' weights calculated from their occurrence in recently learned conflict clauses. Our proposal only affects the variable selection part, and hence it doesn't affect the correctness or completeness of the underlying CDCL algorithm.

4.1 Variable Selection to Satisfy Goals and Subgoals

Our heuristic is based on the following observation: each of the goal literals has to be made *true* by an action, and the precondition literals of each such action have to be made *true* by another action (or they have to be *true* in the initial state.)

The main challenge in defining a variable selection scheme is its integration in the overall SAT solving algorithm in a productive way. To achieve this, the variable selection depends not only on the initial state, the goals and the actions represented by the input clauses, but also the current state of the SAT solver. The state of the solver is primarily characterized by A) the current set of learned clauses and B) the current (partial) valuation reflecting the decisions (variable assignments) and inferences (with unit propagation) made so far. We have restricted the variable selection to use only part B of the SAT solver state, the current partial valuation.

Our algorithm identifies one (sub)goal that is not at the current state of search supported (made true) by an action or the initial state. The search for such support proceeds from the goal literals G at the last time point T in the horizon.

The first step is to find the earliest time point at which a goal literal can become and remain *true*. This happens by going backwards from the end of the horizon to a time point t in which A) an action making the literal *true* is taken or B) the literal is *false* (and it is *true* or *unassigned* thereafter.) The third possibility is that the initial state at time point 0 is reached and the literal is *true* there, and hence nothing needs to be done.

In case A we have an action, and in case B we choose any action that could change the literal from *false* to *true* between t and $t + 1$.³ In case A we find support for the

³ Such an action must exist because otherwise the literal's frame axiom would force the literal *false* also at $t + 1$.

preconditions of the action in the same way. The first action found will be used as the next decision variable in the CDCL algorithm.

The computation is started from scratch at every step of the CDCL procedure because a particular support for a (sub)goal, for example the initial state, may become irrelevant because of a later decision, and a different support needs to be found.

When no (sub)goal without support is found, the current partial assignment represents a plan. The assignment can be made total by assigning the unassigned action variables to *false* and the unassigned fact variables the value they have in the closest preceding time point with a value (inertia).

The algorithm is given in Fig. 2. It takes a stack containing the top-level goals as input, and it returns the empty set or a singleton set $\{o@t\}$ for some action o and time t . Our operation for pushing elements in the stack marks them, and already marked elements are later ignored. This is to prevent subgoal literals occurring in the computation repeatedly. We define $\text{prec}((p, e)) = p$ and $\text{eff}((p, e)) = e$.

```

1: procedure support(Stack, v)
2: while Stack is non-empty do
3:   pop l@t from the Stack;
4:    $t' := t - 1$ ;
5:   found := 0;
6:   repeat
7:     if  $v(o@t') = 1$  for some  $o \in O$  with  $l \in \text{eff}(o)$ 
8:     then
9:       for all  $l' \in \text{prec}(o)$  do push  $l'@t'$  into the Stack;
10:      found := 1;
11:     else if  $v(l@t') = 0$  then
12:        $o := \text{any } o \in O \text{ such that } l \in \text{eff}(o) \text{ and } v(o@t') \neq 0$ ;
13:       return  $\{o@t'\}$ ;
14:      $t' := t' - 1$ ;
15:   until found = 1 or  $t' < 0$ ;
16: end while
17: return  $\emptyset$ ;

```

Fig. 2. Finding support for one unsupported subgoal

Example 1. We illustrate the search for an unsupported (sub)goal and the selection of an action with a problem instance with goals a and b and actions $X = (\{d\}, \{a\})$, $Y = (\{e\}, \{d\})$, and $Z = (\{-c\}, \{b\})$.

variable	0	1	2	3	4	5	6
a	0	0	0	1			
b	0	0	0	1		1	
c	0	0					
d	0	0	0				
e	1						

The timed literals that are considered to be changing from *false* to *true* are shown in boldface. For goal a , the latest time at which a is *false* is 4.

Let's assume that $X@4$ is unassigned, and hence could make a true at 5, and we choose $X@4$ as a support. If this action was already in the partial plan, the precondition d of X would be the next unsupported (sub)goal, and it could be made true by Y at time 2. Hence Y would be the support of d in that case. The precondition e of Y is supported by the initial state and wouldn't need further support.

For the top-level goal b , assume that $Z@2$ is assigned true and hence explains the change of b from false to true between time points 2 and 3. Since Z 's precondition $\neg c$ is satisfied by the initial state, again no further action is required.

4.2 Complexity of the Variable Selection Algorithm

If there are n state variables and the horizon length is T , there are nT variables that represent state variables at different time points. Because each literal is pushed into the stack at most once, the algorithm does the outermost iteration on line 2 for each goal or subgoal at most once, and hence at most nT times in total. The number of iterations of the inner loop starting on line 6 is consequently bounded by nT^2 .

The actual runtime of the algorithm is usually much lower than the above upper bound. A better approximation for the number of iterations of the outer loop is the number of goals and the number of preconditions in the actions in the plan that is eventually found. In practice, the runtime of the CDCL algorithm with our heuristic is still strongly dominated by unit propagation, similarly to CDCL with VSIDS, and the heuristic takes somewhere between 5 and 30 percents of the total SAT solving time.

4.3 Integration in the CDCL Algorithm

Our variable selection scheme is embedded in the CDCL algorithm of Fig. 1 by replacing lines 5 and 6 by the code in Fig. 3. Note that some actions are inferred by unit propagation on line 7 in the CDCL algorithm, and these actions are later handled indistinguishably from actions chosen by the heuristic.

```

1: empty Stack;
2: for all  $l \in G$  do push  $l@T$  into the Stack;
3:  $S := \text{support}(\text{Stack}, v)$ ;
4: if  $S = \{o@t\}$  for some  $o$  and  $t$  then  $v(o@t) := 1$ 
5: else
6:   if there are unassigned  $a@t$  for  $a \in A$  and  $t \geq 1$ 
7:   then  $v(a@t) := v(a@(t - 1))$  for one with minimal  $t$ 
8:   else  $v(o@t) := 0$  for any  $o \in O$  and  $t \geq 0$  with  $o@t$  unassigned;
```

Fig. 3. Variable selection replacing lines 5 and 6 in Fig. 1

The choice of o on line 12 of Fig. 2 and the ordering of the goal literals in the stack on line 3 of Fig. 3 are arbitrary, but for efficiency reasons they must be fixed. In particular, it is important that on line 12 of Fig. 2 the same action is chosen as long as the condition $v(o@t') \neq 0$ condition is satisfied. Intuitively, this is important to avoid losing focus in the CDCL search.

When $\text{support}(\text{Stack}, v) = \emptyset$, all goals and all action preconditions in the current plan are supported by another action or the initial state, and no further actions are needed. The valuation of the SAT solver is usually still partial, because many of the variables corresponding to actions that are not needed are still unassigned, and variables that do not change between two distant time points may be unassigned in some of the intermediate time points. To complete the assignment, we choose, starting from time point 1, unassigned fact variables and assign them the same value they have in the preceding time point (line 7 in Fig. 3). This also sets most of the unassigned action variables *false*. The remaining action variables (usually there are none left) are assigned *false* one by one (line 8 in Fig. 3)

5 Evaluation

We will show that our variable selection heuristic beats VSIDS with *satisfiable* formulas, but not with *unsatisfiable* ones. Further, we will show that our heuristics lead to a planner that is competitive with one of the best planners that don't use SAT.

We used our own SAT solver which is based on CDCL, VSIDS [2], and the phase selection heuristic from RSAT [10]. We tried different clause learning schemes, and it seemed to us that for our problems the Last UIP scheme has a small edge over First UIP [9]. We use the former. We will comment on the relative efficiency of our SAT solver with respect to other solvers later.

As test material we chose 968 problem instances from the biennial international planning competitions from 1998 to 2008. Since our new heuristic is defined for the restricted STRIPS language only, we use all of the STRIPS problems from the planning competitions, except some from the first competition, nor an earlier variant of any benchmark domain that was used in two competitions.

As discussed in Sect. 2, our planner is based on the linear-size \exists -step semantics encoding of Rintanen et al. [5], and the algorithm B with parameter $\gamma = 0.9$. We considered the horizon length parameter $T \in \{0, 5, 10, 15, 20, \dots\}$, and let the planner solve at most 18 SAT instances simultaneously. Most of the problem instances are solved with less than 500 MB of memory, but a couple of dozen required 2 GB or more, up to the 3.5 GB boundary where we could not allocate more memory. The memory restriction was dictated by the 32-bit Ubuntu Linux for which we compiled our programs. All the experiments were run in an Intel Xeon CPU E5405 at 2.00 GHz with a minimum of 4 GB of main memory and using only one CPU core.

To test the performance of the heuristic for both satisfiable and unsatisfiable formulas, with emphasis on the unsatisfiable ones that are important for proofs of optimality of plans, we set up our planner to use the traditional sequential strategy which goes through horizon lengths 0, 1, 2, 3 and so on, until it finds a satisfiable formula. The results of this experiment are summarized in Fig. 4. The plot shows the number of problem instances that are solved (finding a plan) in n seconds or less when using VSIDS and when using the new heuristic (indicated by P). The solver with VSIDS solves about 15 per cent more instances when 300 seconds is spent solving each problem instance.

⁴ Results for the standard parallel plans (\forall -step semantics) and for sequential plans are similar in terms of the improvement our new heuristic yields over VSIDS.

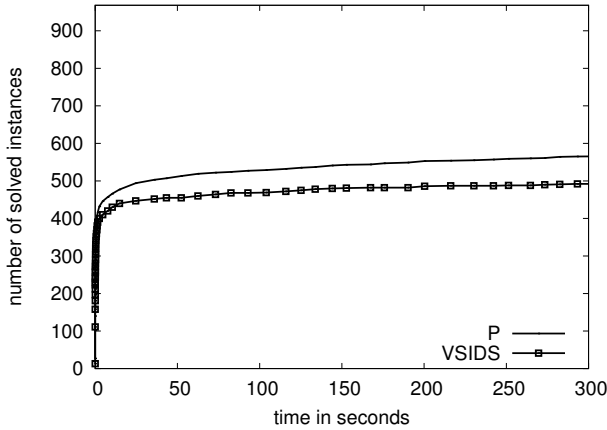


Fig. 4. Number of instances that can be solved in a given time with the sequential strategy

When the optimality proof is required (to show that a plan has the optimal *parallel* length), usually almost all of the effort is spent on the unsatisfiability tests for formulas right below the parallel length of the shortest parallel plan.

When optimality is not required, we use the planner with its default settings, as described earlier, interleaving the solving of several SAT instances for different plan lengths, thus avoiding the completion of the difficult unsatisfiability proofs. In this scenario any improvements in determining satisfiability, i.e. quickly finding a satisfying assignment corresponding to a plan, becomes much more important. The results of this experiment are given in Fig. 5. In this case the new heuristic has a clear advantage over

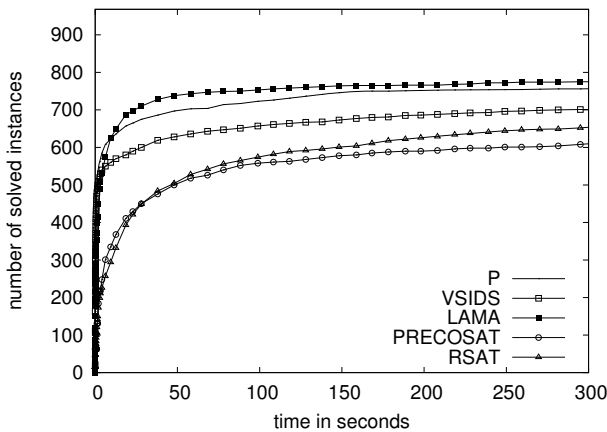


Fig. 5. Number of instances solved by different planners in a given time

Table 1. Number of instances solved in 300 seconds by benchmark domain

	VSIDS	P	LAMA	PRECOSAT	RSAT	
1998-GRIPPER	20	20	20	20	18	20
1998-MPRIME	20	16	18	20	13	13
1998-MYSTERY	19	16	17	19	16	17
2000-BLOCKS	102	71	85	51	52	64
2000-LOGISTICS	76	76	76	76	74	74
2002-DEPOTS	22	21	21	16	21	21
2002-DRIVERLOG	20	15	20	20	18	18
2002-FREECELL	20	4	5	18	5	4
2002-ZENO	20	18	20	20	17	19
2004-AIRPORT	50	40	42	37	22	26
2004-OPTICAL-TELEG	14	14	14	2	14	14
2004-PHILOSOPHERS	29	29	29	BUG	29	29
2004-PIPESWORLD-NOTANK	50	15	20	44	0	0
2004-PSR-SMALL	50	50	49	50	49	50
2004-SATELLITE	36	29	32	30	23	27
2006-PIPESWORLD	50	9	10	38	8	10
2006-ROVERS	40	40	40	40	25	33
2006-STORAGE	30	29	30	18	23	25
2006-TPP	30	26	26	30	30	29
2006-TRUCKS	30	19	29	8	18	19
2008-ELEVATORS	30	13	30	30	16	17
2008-OPENSTACKS	30	15	11	30	12	12
2008-PARCPRINTER	30	30	30	28	30	30
2008-PEGSOLITAIRE	30	25	21	29	30	28
2008-SCANALYZER	30	19	16	27	14	19
2008-SOKOBAN	30	2	4	18	2	2
2008-TRANSPORT	30	10	12	28	1	3
2008-WOODWORKING	30	30	30	28	30	30
total	968	701	757	775	610	653
average number of actions		82.51	61.22	67.13		

VSIDS. The diagram also plots the runtimes of the 2008 International Planning Competition winner LAMA [11]. The performance is almost at the same level, in terms of the number of solved problems in a given time, although the sets of solved instances differ quite a bit. Due to a bug in one of its components LAMA is not able to solve the first instance of OPTICAL-TELEGRAPH and the first 13 instances of PHILOSOPHERS (the rest take longer than 300 seconds.) Some earlier planners have roughly the same performance as LAMA but do much worse with hard problems from the phase transition region or even with easy ones [12].

We also break down the results to different benchmark domains, showing the numbers of problem instances solved in 300 seconds by each planner in Table 1, and show the average number of actions for instances solved by both variants of our planner and

⁵ Following instructions from the authors of LAMA, we decreased the maximum invariant generation time to 60 seconds, to match our time out limit 300 seconds.

LAMA. The numbers of actions in the plans LAMA produces are in general about the same, but for the blocks world problems LAMA's plans are substantially worse.

We also compared our results to the winners of the application/industrial category of the SAT competition in 2009 and 2007, Precosat and RSAT [10]. We translated all the test problems into DIMACS CNF for horizon lengths 0, 10, 20, . . . , 100⁶ and solved them with a 100 second time limit per instance⁷, and then calculated the corresponding Algorithm B runtimes with $\gamma = 0.9$. The Precosat and RSAT runtimes exclude the construction of the CNF formulas and the writing and reading of DIMACS CNF files.

The main reason for the differences between our SAT solver and Precosat and RSAT is preprocessing: for many large SAT instances that we can solve almost immediately, RSAT and Precosat spend considerable time with the preprocessing before starting the search phase. We intend to later investigate better preprocessing strategies for SAT solvers in the planning context. Since the SAT instances for different horizon lengths of one planning instance are closely related, parts of the CNF preprocessing can be shared. This is one of the main avenues to improving preprocessing for planning with SAT.

Precosat is very good with the Peg Solitaire benchmark, presumably because of the preprocessor, but in general it seems that preprocessing in RSAT and Precosat starts to pay off only if there are ten minutes or more available to complete the planning task.

Many of the standard planning benchmarks lead to large SAT problems. The largest SAT problems Precosat solves (within the time bounds explained earlier) are instance 41 of Airport (417476 variables, 92.9 million clauses) and instance 26 of Trucks (926857 variables, 11.3 million clauses). Our planner solves instance 45 of Airport with a completed unsatisfiability test for horizon length 60 (582996 variables and 140.7 million clauses) and a plan for horizon length 70 (679216 variables, 164.1 million clauses). Our planner, with the new heuristic, also solves instance 34 of Satellite, with a plan found for horizon length 40 (8.5 million variables, 29.9 million clauses) backtrack-free in 20.01 seconds excluding translation into SAT and including effort to find unsatisfiability proofs for shorter horizon lengths. These are extreme cases. Most of the instances have less than 1 million variables and at most a couple of million clauses.

6 Related Work

6.1 Earlier Planning Algorithms

Many earlier algorithms add actions to incomplete plans to support an existing goal or subgoal, for example the partial-order planning algorithms [13]. The LPG planner [14] does stochastic local search in the space of incomplete plans with parallel actions similar to the SAT-based approach. LPG's choice of actions to be added in the current incomplete plan is based on the impact of the action on violations of constraints describing the solutions.

Heuristic search [15] has long been an essential for problem solving in AI, but its use in planning was limited until the ground-breaking work of Bonet et al. [16]. Research

⁶ We use the step 10 to reduce the time to perform the experiment. This affects the runtimes negligibly. For the blocks world problems we used horizon lengths up to 200.

⁷ This is more than enough to determine the planners' runtimes up to time out limit 300 seconds.

quickly focused on state-space search guided by heuristics derived from declarative problem descriptions in a generic, problem-independent manner. A main emphasis in the research on classical planning has been in finding better heuristics, with very limited efforts to try something else than state-space search.

The Graphplan algorithm [17] uses backward search, constrained by the planning graph structure which represents approximate (upper bound) reachability information. The action selection of GraphPlan's search may resemble our action selection: given a subgoal l at time t , the choice of an action to reach l is restricted to actions in the planning graph at level $t - 1$. This same constraint on action selection shows up in any extraction of action sequences from exact distance information, for example in BDD-based planning [18] and corresponding model-checking methods. However, the data structures representing the distances (the planning graph or the BDDs) are not used as a heuristic as in our work: when the action choice for achieving l is not restricted by the contents of the planning graph, Graphplan will choose an arbitrary action with l as an effect. Another major difference is of course that our heuristic leverages on the search state of the CDCL algorithm (the learned clauses). This is the main reason why our heuristic, despite its extreme simplicity, is more informative than the more complex heuristics earlier used in AI planning.

6.2 Earlier Planners That Use SAT

The best known planner that uses SAT is BLACKBOX by Kautz and Selman [19]. Rintanen et al. [5] demonstrate that their \forall -step semantics encoding is often substantially faster than the BLACKBOX encoding, sometimes by a factor of 20 or more. Both encodings use the same definition of parallel plans. Runtime data in Sideris and Dimopoulos [20] indicates that newer planners in the BLACKBOX family implement slower encodings than BLACKBOX. For example, SATPLAN06 is often twice as slow as BLACKBOX. The only other encoding that is comparable to the \forall -step semantics encoding of Rintanen et al. in terms of efficiency and size is the recent factored encoding of Robinson et al. [21].

The more relaxed notion of parallel plans used in our planner, the \exists -step semantics [5,22], allows shorter horizons and smaller formulas, and therefore leads to substantial efficiency improvements. This and parallelized search strategies [6] often mean further one, two or more orders of magnitudes of speed up over other SAT-based planners.

6.3 Domain-Specific Heuristics for SAT Solving

Not much is known about using problem specific heuristics in SAT solving or the workings of SAT solvers when solving planning problems. Beame et al. [8] demonstrate the utility of a problem-specific variable selection heuristic for a clause-learning algorithm solving a combinatorial problem (pebbling formulas.) They demonstrate an improvement in finding unsatisfiability proofs.

The decision variable heuristic proposed in this paper focuses on action variables, and only assigns fact variables at the last stages to complete the assignment. Theoretical

results indicate that the efficiency of CDCL is decreased if variable assignments are restricted to a subset of the variables only, even if those variables are sufficient for determining satisfiability and unsatisfiability [23]. It is not clear to us what the practical or theoretical implications of these results are in our setting, more specifically because planning and state-space reachability problems are only a subset of all satisfiability problems. The experiments of Järvisalo and Junttila with instances from bounded-model checking style deadlock detection and LTL model-checking suggest that a CDCL solver with VSIDS restricted to a subset of decision variables fares worse than VSIDS without the restriction. In contrast, we have a heuristic that has such limitations, and the unlimited decision heuristic (VSIDS) with our test material fares in general worse. Of course, the results of Järvisalo and Junttila could be seen as saying that there are more effective variants of our heuristic which sometimes also choose fact variables as decision variables. Further, all known restrictions on SAT solving efficiency (in a given proof system) apply to unsatisfiability proofs only, which are not the focus of our work.

7 Conclusions and Future Work

The contribution of this paper is a simple yet powerful variable selection strategy for clause-learning SAT solvers that solve AI planning problems, as well as an empirical demonstration that the strategy outperforms VSIDS for standard planning benchmarks when no proofs of the optimality of the horizon length is required. A main additional benefit over VSIDS is that the variable selection strategy is understandable in terms of the planning problem, and that there are obvious and intuitive avenues for developing it further. The basic variable selection strategy is particularly promising because the features that makes it strong are largely complementary to the important features of VSIDS, making it possible to combine them, for example by adopting the weights of literals from VSIDS to the planning heuristic. This is a focus of future work, as is finding ways of doing the unsatisfiability proofs more efficiently.

Immediate improvement opportunities arise for example from the possibility of changing the order in which the top-level goals and the preconditions of an action are considered. Our initial experimentation in this area has demonstrated that the approach can be dramatically strengthened further, leading to planners that substantially outperform modern planners such as LAMA.

The main ideas in this work are quite general, and could be easily adapted to other applications of SAT and constraint-satisfaction, for example model-checking [24] and diagnosis [25], and of more expressive logics, such as QBF [26].

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. We thank Hector Geffner and Patrik Haslum for comments on earlier versions of this paper.

References

1. Kautz, H., Selman, B.: Planning as satisfiability. In: Neumann, B. (ed.) *Proceedings of the 10th European Conference on Artificial Intelligence*, pp. 359–363. John Wiley & Sons, Chichester (1992)
2. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC 2001)*, pp. 530–535. ACM Press, New York (2001)
3. Bonet, B., Geffner, H.: Planning as heuristic search. *Artificial Intelligence* 129(1-2), 5–33 (2001)
4. Cook, S.A.: The complexity of theorem proving procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158 (1971)
5. Rintanen, J., Heljanko, K., Niemelä, I.: Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13), 1031–1080 (2006)
6. Rintanen, J.: Planning and SAT. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) *Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications*, vol. 185, pp. 483–504. IOS Press, Amsterdam (2009)
7. Rintanen, J.: Regression for classical and nondeterministic planning. In: Ghallab, M., Spyropoulos, C.D., Fakotakis, N. (eds.) *ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence*, pp. 568–571. IOS Press, Amsterdam (2008)
8. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research* 22, 319–351 (2004)
9. Mitchell, D.G.: A SAT solver primer. *EATCS Bulletin* 85, 112–133 (2005)
10. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) *SAT 2007. LNCS*, vol. 4501, pp. 294–299. Springer, Heidelberg (2007)
11. Richter, S., Helmert, M., Westphal, M.: Landmarks revisited. In: *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-2008)*, pp. 975–982. AAAI Press, Menlo Park (2008)
12. Rintanen, J.: Phase transitions in classical planning: an experimental study. In: Zilberstein, S., Koehler, J., Koenig, S. (eds.) *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, pp. 101–110. AAAI Press, Menlo Park (2004)
13. McAllester, D.A., Rosenblitt, D.: Systematic nonlinear planning. In: *Proceedings of the 9th National Conference on Artificial Intelligence*, vol. 2, pp. 634–639. AAAI Press/The MIT Press (1991)
14. Gerevini, A., Serina, I.: Planning as propositional CSP: from Walksat to local search techniques for action graphs. *Constraints Journal* 8, 389–413 (2003)
15. Pearl, J.: *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, Reading (1984)
16. Bonet, B., Loerincs, G., Geffner, H.: A robust and fast action selection mechanism for planning. In: *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI 1997)*, pp. 714–719. AAAI Press, Menlo Park (1997)
17. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2), 281–300 (1997)
18. Cimatti, A., Giunchiglia, E., Giunchiglia, F., Traverso, P.: Planning via model checking: a decision procedure for \mathcal{AR} . In: Steel, S., Alami, R. (eds.) *ECP 1997. LNCS*, vol. 1348, pp. 130–142. Springer, Heidelberg (1997)

19. Kautz, H., Selman, B.: Unifying SAT-based and graph-based planning. In: Dean, T. (ed.) *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pp. 318–325. Morgan Kaufmann Publishers, San Francisco (1999)
20. Sideris, A., Dimopoulos, Y.: Constraint propagation in propositional planning. In: *ICAPS 2010. Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pp. 153–160. AAAI Press, Menlo Park (2010)
21. Robinson, N., Gretton, C., Pham, D.N., Sattar, A.: SAT-based parallel planning using a split representation of actions. In: Gerevini, A., Howe, A., Cesta, A., Refanidis, I. (eds.) *ICAPS 2009. Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, pp. 281–288. AAAI Press, Menlo Park (2009)
22. Wehrle, M., Rintanen, J.: Planning as satisfiability with relaxed \exists -step plans. In: Orgun, M.A., Thornton, J. (eds.) *AI 2007. LNCS (LNAI)*, vol. 4830, pp. 244–253. Springer, Heidelberg (2007)
23. Järvisalo, M., Junttila, T.: Limitations of restricted branching in clause learning. *Constraints Journal* 14, 325–356 (2009)
24. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) *TACAS 1999. LNCS*, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
25. Grastien, A., Anbulagan, R.J., Kelareva, E.: Diagnosis of discrete-event systems using satisfiability algorithms. In: *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pp. 305–310. AAAI Press, Menlo Park (2007)
26. Rintanen, J.: Asymptotically optimal encodings of conformant planning in QBF. In: *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI 2007)*, pp. 1045–1050. AAAI Press, Menlo Park (2007)

Value-Ordering Heuristics: Search Performance vs. Solution Diversity

Yevgeny Schreiber

Intel Corporation, Haifa, Israel
yevgeny.schreiber@intel.com

Abstract. We examine the behavior of dynamic value-ordering heuristics in a CSP under the requirement to generate a large number of diverse solutions as fast as possible. In particular, we analyze the trade-off between the solution search performance and the diversity of the generated solutions, and propose a general probabilistic approach to control and improve this trade-off. Several old/new learning-reuse heuristics are described, extending the survivors-first value-ordering heuristics family. The proposed approach is illustrated on a real-world set of examples from the Automatic Test Generation problem domain, as well as on several sets of random binary CSPs.

Keywords: Automatic Test Generation, Learning Reuse, Solution Diversity, Survivors-First, Value Ordering CSP Heuristic.

1 Introduction

A Constraint Satisfaction Problem (CSP) is defined as a set $\mathcal{X} = \{x_1, \dots, x_n\}$ of variables, a set $\mathcal{D} = \{d_1, \dots, d_n\}$ of the corresponding value domains, and a set $\mathcal{C} = \{c_1, \dots, c_m\}$ of constraints that define the allowed value combinations of the variables. A CSP solution is an assignment of values to variables so that all the constraints in \mathcal{C} are satisfied.

The general solution search method can be described as follows: (a) Repeatedly select an unassigned variable $x \in \mathcal{X}$, (b) try to assign to x one of the remaining values in its domain, and (c) propagate every constraint $c \in \mathcal{C}$ that involves x , by removing the conflicting values from the domains of all the other variables that are involved in c . If, as a result of the constraint propagation, a variable domain becomes empty, the search backtracks, the value that has been assigned to x is removed from its domain, and all the domains that have been reduced after the last assignment of x are restored.

A value-ordering heuristic determines which value is selected by the algorithm at step (b). If the problem is solvable, then, on average, a solution can be reached more quickly if the selected value maximizes the number of possibilities to assign variables that are still unassigned (that is, has a little chance of leading to a conflict). Various existing methods use this heuristic and attempt to evaluate, a priori, a possible search-space reduction effect of an assignment. For example, Geelen's Promise [2] computes for each value the product of the numbers of

supports in the domains of the unassigned variables. Refalo [14] suggested a method based on learning the results of previous assignments on one hand and on look-ahead techniques on the other hand. Another family of value-ordering heuristics was introduced by Zhang and Epstein [17]. These heuristics, called *survivors-first*, use simple statistics accumulated during the search to select the value that has been involved in less conflicts than others. The survivors-first heuristics are easy to implement, their run-time and memory overhead is low, and, as it is shown in [17], they can be quite helpful in many cases.

However, there is a class of applications where the cost of using these and other value-ordering heuristics comprises an additional factor. Sometimes it is required to find *several* solutions of a given problem; moreover, these solutions should often be as *different* from each other as possible. The latter requirement is called *solution diversity*. Hebrard et al [5] distinguish between *off-line* diversity/similarity problems where the whole required set of solutions is computed at once, and their *on-line* counter-part where the solutions are computed incrementally. The MAXDIVERSE k SET problem of computing k (approximately) maximally diverse solutions of a given CSP is defined and studied in [5]; the approach is demonstrated on $k = 3$. In particular, the authors compare a random value shuffle with a value-ordering heuristic that first selects values that have been least used in previous solutions.

In this paper we concentrate on the on-line version of the problem, with the following motivation. The solution diversity requirement is very important in the Automatic Test Generation Problem (ATGP) (see [4,6,11,13,15]). The problem is to generate automatically a valid test for a given specification (usually, of some hardware), which is a sequence of a large number of instructions. These instructions have to be diverse, in order to trigger as many as possible different hardware events. Van Hentenryck et al [6] propose to solve this MAXDIVERSE k SET ATGP using the constraint-based local search techniques, and demonstrate their solution on a (small scale) ATGP; in particular, they mention the trade-off between the solution diversity and the performance of their algorithm.

We continue and extend the above lines of work in the following direction. The computation time is usually very important for ATGP, since thousands of tests have to be generated even for a small subset of a modern hardware specification. Moreover, when measuring the quality of the tests, it often makes sense to consider the overall quality of the whole test-base; therefore generating a small number of high-quality (diverse) tests can be less useful than generating a large number of somewhat less diverse tests. We can therefore talk about a *trade-off* between the search performance and the solution diversity.

Since the computation time is important, we concentrate on the case where it is too expensive to look for an optimal solution of the MAXDIVERSE k SET — rather, we are interested in an approximation whose computational cost is as close as possible to the cost of k independent searches for a single solution. To reach this goal we use value-ordering heuristics. First, we analyze the trade-off between the search performance and the solution diversity, as a function of the value-ordering heuristic. Second, we suggest general methods to characterize and

extend value-ordering heuristics so that this trade-off could be easily controlled. Finally, we show that several value-ordering heuristics can obtain a relatively good performance/quality trade-off.

Before we continue, note the difference between solution diversity and *solution distribution*. There are several works (see e.g. [13]) that provide methods for generating a uniformly random sample of solution space. The goal there is to produce each solution with probability that is as close as possible to $\frac{1}{N}$, where N is the total number of solutions. However, the uniform distribution does not necessarily lead to a sample of solutions that are highly different from each other. For example, consider a solution space composed of a small subset S_1 of solutions where each variable is assigned a different value, and a much larger set of solutions S_2 so that only a single variable is assigned a different value in each solution. Then there is a high probability that most of the produced solutions in a uniformly random sample would belong to S_2 rather than to S_1 . Obviously, the diversity of the produced solutions would be relatively low.

The rest of the paper is organized as follows. In the next section we analyze the relation between value-ordering heuristics and the diversity of the produced solution set. In Section 3 we describe several simple value-ordering heuristics that generalize and extend the *survivors-first* heuristics family introduced in [17] by using a few simple techniques that allow to control and improve the trade-off between the search performance and the solution diversity. This approach is illustrated in Section 4 on random binary CSPs and on a full scale ATGP involving thousands of variables and constraints, where many variables have huge domains. In Section 5 we summarize the discussion.

2 Analysis

Given a CSP P , the MAXDIVERSE k SET problem is defined in [5] as computing a set S of k solutions of P that maximizes $\delta(S) = \sum_{s, s' \in S} \delta(s, s')$, where $\delta(s, s')$ measures the distance between a pair of solutions.

It is often convenient to use the Hamming distance as δ . That is, given $s = \langle s_1, \dots, s_n \rangle$ and $s' = \langle s'_1, \dots, s'_n \rangle$, we define $H_i(s, s') = 1$ if $s_i \neq s'_i$ and $H_i(s, s') = 0$ otherwise, for $1 \leq i \leq n$. Then $\delta(s, s')$ is defined as $\sum_{i=1}^n H_i(s, s')$. Note that $\delta(S)$ can be normalized to remove the dependence on n and k by dividing it by $\frac{nk(k-1)}{2}$.

For the purpose of the analysis in this paper we assume that δ is the Hamming distance; however, it is also possible to apply a similar approach for other metrics as well (e.g. the more general *weighted* Hamming distance, where each $H_i(s, s')$ is multiplied by a respective weight, etc).

Consider a probabilistic search algorithm $A_{\mathcal{H}}$ that selects the values at the search tree nodes according to some value-ordering heuristic \mathcal{H} ; $A_{\mathcal{H}}(P)$ stops when a solution of P is found or until it reaches the conclusion that P does not have a solution. Given a solution $s = \langle s_1, \dots, s_n \rangle$ found by $A_{\mathcal{H}}(P)$, let s_i be the value assigned to $x_i \in \mathcal{X}$ in s , and let u be a value in d_i . Then $p(s_i = u)$ is defined to be the probability that $s_i = u$. Denote by $\tilde{d}_i \subseteq d_i$ the set of values u

for which $x_i = u$ is part of at least one solution of P . (Obviously, $p(s_i = v) = 0$ for each $v \in d_i \setminus \tilde{d}_i$.)

Definition 1. *The uniform value distribution property is the set of equalities $p(s_i = u) = \frac{1}{|d_i|}$ for each $x_i \in \mathcal{X}$ and each $u \in \tilde{d}_i$.*

Lemma 1. *If the uniform value distribution property holds, then the expected value $E[\delta(S)]$ for a random set S of solutions found by $A_{\mathcal{H}}(P)$ is maximal.*

Proof. Since $E[\delta(S)] = \sum_{s,s' \in S} E[\delta(s, s')]$, and since $E[\delta(s, s')]$ is equal for each pair of random solutions in S , it is sufficient to show that when the uniform value distribution property holds, $E[\delta(s, s')] = \sum_{i=1}^n E[H_i(s, s')]$ is maximal.

Since it is given that s and s' are solutions of P , we have

$$E[H_i(s, s')] = \sum_{u \in \tilde{d}_i} p(s_i = u)(1 - p(s_i = u)) = \sum_{u \in \tilde{d}_i} p(s_i = u) - \sum_{u \in \tilde{d}_i} (p(s_i = u))^2 .$$

Using the fact that $\sum_{u \in \tilde{d}_i} p(s_i = u) = 1$ and $0 < p(s_i = u) \leq 1$ for each $u \in \tilde{d}_i$, it is not difficult to see that this function reaches the maximum of $1 - \frac{1}{|d_i|}$ when the uniform value distribution property holds. (To see this, define a uniformly random variable X whose possible values are p_1, \dots, p_m so that $\sum_i^m p_i = 1$; then the variance of X equals to $\frac{1}{m} \sum_i^m p_i^2 - \frac{1}{m^2}$, which equals to 0 when each $p_i = \frac{1}{m}$.) As a consequence, $\sum_{i=1}^n E[H_i(s, s')]$ reaches the maximum of $n - \sum_{i=1}^n \frac{1}{|d_i|}$. \square

Note that Lemma 1 neither shows how to reach the uniform value distribution property, nor even states that this property is possible for every P .

Let σ_j be a partial assignment $\langle x_1 = s_1, \dots, x_j = s_j \rangle$, for $j < n$, and denote by $p(x_i = u \mid \sigma_{i-1})$ the probability of $A_{\mathcal{H}}(P)$ to assign u to x_i at the search node where the variables x_1, \dots, x_{i-1} are assigned according to σ_{i-1} . By Lemma 1, in order to maximize $E[\delta(S)]$, we would like to set all the probabilities $p(x = u \mid \sigma)$ so that the uniform value distribution property would hold. Unfortunately, it is an NP-hard problem [8]. There exist several approximation methods [9,12] that can deal with the so called *distribution constraints*, in which the desired distribution of values of some variables in the solution can be specified. However, such methods are not applicable here since they can usually deal with only a limited number of distribution constraints, while we would like the uniform value distribution property to hold for each $x_i \in \mathcal{X}$.

Another approximation method attempts to maximize $E[\delta(S)]$ *interactively*, as follows. Let S' be a set of already produced $k' < k$ solutions. Denote by $\tilde{p}(s_i = u)$ an approximation of $p(s_i = u)$, evaluated according to the behavior of $A_{\mathcal{H}}(P)$ during the k' searches. Then the uniform value distribution property can be approximated by the condition $\tilde{p}(s_i = u) = \tilde{p}(s_i = v)$ for each $u, v \in d_i$. Note that we use d_i instead of \tilde{d}_i in this approximation, since \tilde{d}_i is unknown a priori. However, selecting a value in $d_i \setminus \tilde{d}_i$ does not affect the resulting value of $E[\delta(S)]$ if the search is not limited and it runs until a solution is found.

Each term of the sum $\sum_{i=1}^n E[H_i(s, s')]$ can be greedily maximized independently, by assigning to x_i a value u whose $\tilde{p}(s_i = u)$ is minimal in d_i . A variant of this approach where u participates in the smallest number of solutions with $s_i = u$ in S' was used in [6] (among other heuristics); we describe a few additional variants of this approach in Section 3. When a new solution is added to S' , all the evaluated approximations are updated.

Unfortunately, such greedy maximization attempt of each $E[H_i(s, s')]$ does not necessarily lead to a maximization of $\sum_{i=1}^n E[H_i(s, s')]$, since it can decrease several other terms of the sum, and so there is no guarantee that the resulting $E[\delta(S)]$ would be higher than if using a different value ordering heuristic. Moreover, attempts to maximize $E[\delta(S)]$ are likely to increase the probability of selection of values whose probability to survive until the end of the search are relatively low, and therefore to increase the runtime of the search.

We verify the correctness of this analysis by experiments that are described in Section 4. Before that, in the next section we describe several value-ordering heuristics and practical techniques that provide some control over the trade-off between the performance and the solution diversity.

3 Value-Ordering Heuristics

The most basic value-ordering heuristic that comes to mind when considering solution diversity is the RANDOM heuristic, which simply selects a uniformly random value from d_i .¹ It is observed in [5] and confirmed by our experiments described in the next section that this simple heuristic often achieves relatively high solution diversity in problem instances that have many possible solutions (which is usually the interesting case when considering the solution diversity requirement), and that in such problems RANDOM performs relatively fast.

This observation can be explained by the fact that in such problems there is usually a relatively large number of values in every domain whose selection does not lead to a conflict, and therefore there is a relatively high probability for RANDOM to select a value without paying a high performance penalty for a uniformly random selection. Moreover, the procedure itself of randomly selecting a value from a domain is very fast. Still, by using the following learning-reuse value ordering heuristics we attempt to achieve a better trade-off between performance and solution diversity than RANDOM does.

In the following definitions of value ordering heuristics \mathcal{H} we denote by $p(x_i, v)$ the probability for a random algorithm $A_{\mathcal{H}}$ to select the value v at a search node where the variable x_i is assigned.

1. LEASTFAILS (similar to RVO in [17]) assigns x_i a value v with the lowest recorded number $b(x_i, v)$ of attempts to assign v to x_i that have led to a conflict (and therefore to a removal of v from the domain of x_i); if there are k such values in d_i , then $p(x_i, v) = \frac{1}{k}$ for each such v , and $p(x_i, v) = 0$ for every other value in d_i .

¹ Here and below d_i denotes the “current” subset of values from the original domain that have not been removed by the constraint propagation process so far.

2. **BESTSUCCESSRATIO** (similar to RSVO in [17]) assigns x_i a value v with the lowest ratio of $b(x_i, v)$ over the total recorded number $a(x_i, v)$ of attempts to assign v to x_i ; if there are k such values v in d_i , then $p(x_i, v) = \frac{1}{k}$.
3. **PROBLEASTFAILS** is a probabilistic version of **LEASTFAILS**: it assigns x_i a value v with a probability that is inversely proportional to $b(x_i, v)$; that is, $p(x_i, v) = \frac{B}{b(x_i, v)}$, where $B = \sum_{u \in d_i} \frac{1}{b(x_i, u)}$ does not depend on v . (We assume here that $b(x_i, v) > 0$; the case with $b(x_i, v) = 0$ is discussed further in this section.)
4. **PROBBESTSUCCESSRATIO** is a probabilistic version of **BESTSUCCESSRATIO**: it assigns x_i a value v with the a probability that is proportional to $1 - \frac{b(x_i, v)}{a(x_i, v)}$; that is, $p(x_i, v) = \frac{a(x_i, v) - b(x_i, v)}{(|d_i| - C)a(x_i, v)}$, where $C = \sum_{u \in d_i} \frac{b(x_i, u)}{a(x_i, u)}$ does not depend on v . (We assume here that $a(x_i, v) > 0$; the case with $a(x_i, v) = 0$ is discussed below.)
5. **PROBMOSTFAILS** (extending a related heuristic used in [5]) assigns x_i a value v with a probability that is proportional to $b(x_i, v)$; that is, $p(x_i, v) = \frac{b(x_i, v)}{\sum_{u \in d_i} b(x_i, u)}$ (assuming that $\sum_{u \in d_i} b(x_i, u) > 0$).
6. **PROBWORSTSUCCESSRATIO** (extending a related heuristic used in [5]) assigns x_i a value v with a probability that is proportional to $\frac{b(x_i, v)}{a(x_i, v)}$; that is, $p(x_i, v) = \frac{b(x_i, v)}{Ca(x_i, v)}$ (as above, assuming that $a(x_i, v) > 0$).

According to the analysis in Section 2, the heuristics **PROBMOSTFAILS** and **PROBWORSTSUCCESSRATIO** greedily attempt to maximize each $E[H_i(s, s')]$ interactively in order to get a better Hamming distance between the generated solutions. (Note that here we only use the probabilistic versions of such heuristics, since always selecting a value which usually leads to a conflict is expected to lead to an unbearable performance penalty.) The heuristics **LEASTFAILS** and **BESTSUCCESSRATIO** attempt to find a solution as fast as possible, and **PROBLEASTFAILS** and **PROBBESTSUCCESSRATIO** try to stay in the middle, sacrificing some of the solution diversity for the sake of performance.

We observe two factors that can affect the trade-off between the solution diversity and the search performance of each of the above value-ordering heuristics, within the framework of the general policy of the heuristic. We call these factors the *conservativeness* and the *aggressiveness*.

The conservativeness can only be considered for a heuristic that does not necessarily determine a full order between all the values in a domain. For example, the above *learning-reuse* heuristics can only order values that have already been attempted to be assigned to variables in the past. In this and other cases it must be decided what $p(x_i, u)$ should be used for the values $u \in d_i$ whose order cannot be currently determined by the heuristic. The more conservative a heuristic \mathcal{H} is, the lower is the probability $p(x_i, u)$ to select such an “unordered” value. We can define the conservativeness of a value-ordering heuristic \mathcal{H} separately for each x_i whose domain contains at least one unordered value $u \in d_i$ as $C_i(\mathcal{H}) = 1 - p(x_i, u)$. A low $C_i(\mathcal{H})$ can lead to a more random behavior in the first few times that x_i is assigned, which can sometimes result in an inferior search

performance. However, this allows to prevent situations where many unordered values are never selected and therefore never become ordered. It is natural to control $C_i(\mathcal{H})$ using a parameter α that represents an “initial score” of an unordered value u : for LEASTFAILS, PROBLEASTFAILS, and PROBMOSTFAILS this can be some initially expected value of $b(x_i, u)$, and for BESTSUCCESSRATIO, PROBBESTSUCCESSRATIO, and PROBWORSTSUCCESSRATIO, α can represent some initially expected ratio $b(x_i, u)/a(x_i, u)$.

The aggressiveness of a heuristic is a measure of non-uniformness of $p(x_i, v)$ over all $v \in d_i$. It can be defined separately for each $x_i \in \mathcal{X}$ as

$$A_i(\mathcal{H}) = \max_{v \in d_i} \{p(x_i, v)\} - \min_{v \in d_i} \{p(x_i, v)\} .$$

(Note that there are more than one way to define the measure of non-uniformness of $p(x_i, v)$; we have selected one that is convenient for analysis.) Here we assume that for every $v \in d_i$ some $p(x_i, v)$ is set, including unordered values whose $p(x_i, v)$ is set according to $C_i(\mathcal{H})$. The less aggressive a heuristic is, the more similar to RANDOM it behaves. For the heuristics PROBLEASTFAILS, PROBBESTSUCCESSRATIO, PROBMOSTFAILS, and PROBWORSTSUCCESSRATIO it is natural to control $A_i(\mathcal{H})$ as follows: The distribution of $p(x_i, v)$ over all $v \in d_i$ can be smoothed using a parameter $\beta > 0$, or amplified using $-\min_{v \in d_i} \{p(x_i, v)\} < \beta < 0$:

$$p_{\text{new}}(x_i, v) = \frac{p(x_i, v) + \beta}{1 + \beta |d_i|}, \text{ so the new aggressiveness is : } \frac{A_i(\mathcal{H})}{1 + \beta |d_i|} .$$

Note that setting $\beta > 0$ allows to avoid all the cases in the definition of these heuristics where $b(x_i, v) = 0$ or $a(x_i, v) = 0$.

For the heuristics LEASTFAILS and BESTSUCCESSRATIO, which cannot select a value whose score is not the highest in the domain, a *tie-range* parameter $0 \leq \gamma \leq 1$ can be used to smooth differences between “sufficiently close” values. (In [17] a constant $\gamma = 0.05$ is used in a related context.) Let d'_i be the set of all the values $v \in d_i$ so that $(1 + \gamma)p(x_i, v) \geq \max_{u \in d_i} p(x_i, u)$. Then for each v in d'_i : $p_{\text{new}}(x_i, v) = \frac{1}{|d'_i|} \sum_{u \in d'_i} p(x_i, u)$. Let $u' \in d_i$ be the value whose $p(x_i, u')$ is minimal. If $p_{\text{new}}(x_i, u') = p(x_i, u')$ then $p_{\text{new}}(x_i, u') = p(x_i, u') = 0$, and therefore $\frac{A_i(\mathcal{H})}{1 + \gamma} \leq A_i(\mathcal{H})_{\text{new}} \leq A_i(\mathcal{H})$; otherwise the new aggressiveness $A_i(\mathcal{H})_{\text{new}} = 0$, and the resulting behavior is similar to RANDOM.

Note that all the learning-reuse heuristics, including the six heuristics described above, collect information during the search and update the probabilities $p(x, v)$ on the fly. As a result, if the parameters β, γ that control $A_i(\mathcal{H})$ are only set once at the beginning of the search, the aggressiveness of \mathcal{H} grows as the search progresses. Moreover, during the search, as more and more initially unordered values become ordered by the heuristic, the conservativeness of \mathcal{H} plays less and less important part. Note also that, as usual with learning-reuse heuristics, no information is available at the beginning of the search for a first solution. As a consequence, all the values in each domain d_i are initially unordered, $p(x_i, u) = \frac{1}{|d_i|}$ for each $u \in d_i$, and $A_i(\mathcal{H}) = 0$, which means that each above heuristic in the beginning of the first search is equivalent to RANDOM.

3.1 Adaptive Aggressiveness

There is an additional conclusion that can be made from the analysis in Section 2 and from the fact that the computation of Hamming distance is symmetric with respect to the element order in the compared vectors. The performance/diversity trade-off can be improved as follows, when it is possible to identify a subset $\mathcal{X}' \subset \mathcal{X}$ of variables that form a sub-problem that (a) is difficult to satisfy, and (b) is quite disconnected from the rest of the variables (that is, assigning variables in \mathcal{X}' does not cause “much” domain reduction for variables outside \mathcal{X}'). We could make the behavior of the heuristic more performance-oriented for variables in \mathcal{X}' , and more diversity-oriented for the rest of the variables. However, it is difficult to identify such relatively independent subsets of variables (characteristic (b)) without applying additional techniques, which are out of scope of this paper and are left for future research. On the other hand, characteristic (a) can be approximated quite easily, using the values of $a(x_i, v)$ and $b(x_i, v)$ that are collected by the learning-reuse techniques described above.

We can therefore define, for each of the six learning-reuse heuristics \mathcal{H} described above, an *adaptive* version \mathcal{H}_{ad} that corrects during the search the initially given $A_i(\mathcal{H})$, individually for each $x_i \in \mathcal{X}$, according to the estimation $r(x_i)$ of the average $\frac{1}{b(x_i, v)}$ (for all the heuristics that do not consider $a(x_i, v)$) or $\frac{a(x_i, v)}{b(x_i, v)}$ (for the heuristics that consider both $a(x_i, v)$ and $b(x_i, v)$) over all $v \in d_i$. In particular, the adaptive versions of LEASTFAILS and BESTSUCCESSRATIO (PROBLEASTFAILS and PROBBESTSUCCESSRATIO) use the parameter γ (respectively, β) to increase $A_i(\mathcal{H})$ for x_i with low $r(x_i)$. This results in a more uniform selection of values from a domain of a variable that is not involved in “tight” constraints than for a variable whose assignments often fail. On the other hand, the adaptive versions of PROBMOSTFAILS and PROBWORSTSUCCESSRATIO use β to *decrease* $A_i(\mathcal{H})$ for x_i with low $r(x_i)$, to make the heuristic less aggressive for variables whose assignments often fail.

4 Experiments

We ran the experiments on the following two sets of problems:

(1) Randomly generated problems, with the $\langle a, b, c, d \rangle$ notation: a is the number of variables, b is the domain size of each variable, c is the number of binary constraints, and d is the number of incompatible value pairs in each constraint. We used $\langle 50, 10, 225, 35 \rangle$ and $\langle 50, 10, 225, 37 \rangle$, with 30 problems in each set; we have also tested our results on $\langle 50, 10, 450, 20 \rangle$ and the results are quite similar, but are not shown here due to space limitations. We looked for 30 solutions of each problem and measured the Hamming distance between each pair of solutions. Note that the constraints in $\langle 50, 10, 225, 37 \rangle$ are tighter than those in $\langle 50, 10, 225, 35 \rangle$, and as a result these problems are much harder, but both sets of problems are still not in the transition phase [16], and therefore have many solutions. (We are not interested in transition phase problems in our experiments since the question of solution diversity is much less relevant there.) All the random problems were

solved using the variable ordering heuristic that always picked the variable with the currently minimal domain.

(2) Automatic Test Generation problems. Each ATGP involves thousands of variables and constraints that model Intel 64 and IA-32 processor architecture [7]. Additionally, each ATGP contains a small set of constraints that cause the generated instruction sequence (the solution of ATGP) to trigger a specific set of interesting events when executed on a corresponding processor. In general, the goal is to find a set of solutions (test instructions) that (a) satisfy all these constraints and (b) are as different from each other as possible. Here we use the $\langle n, m \rangle$ notation: n is the number of problems in the set, and m is the average number of instructions that is required to generate for each problem. We experimented with two sets of ATGP problems: $\langle 15, 21 \rangle$ and $\langle 26, 7 \rangle$; we have also tested our results on $\langle 15, 41 \rangle$ and the results are similar, but are not presented here. The set $\langle 26, 7 \rangle$ was added to the experiment in order to test the behavior of the heuristics with only a relatively small number of required solutions.

Except the complexity of the problems, there are many additional differences between the ATGP and the random CSP benchmark; we mention here two differences that are relevant for our experiments. First, even when all the constraints are identical for each ATGP instruction, computing a sequence of n instructions is not quite equivalent to finding n solutions for the same problem. The reason is that each instruction in the sequence modifies the architectural state of the processor, which is modeled as part of the CSP. As a result, the problem is a little different each time that we are looking for a new solution, which reduces the efficiency of the learning-reuse heuristics. (See [4,15] for a description of methods that deal with the architectural state change in ATGP.) Another difference is the computation of distance between the ATGP solutions. ATGP is a complicated problem, where the *existence* of many variables depends on values of other variables, and is therefore modeled as Conditional CSP (CCSP, or Dynamic CSP in [10]); see [6,11] for a detailed description of this complication and of methods to compute distance between solutions of a CCSP.

4.1 Results

We compare the RANDOM heuristic with each of the heuristics described above (six basic and six adaptive versions). Each of the twelve heuristics is used with nine sets of parameters that control its aggressiveness and conservativeness. In total, each problem in the test bench is solved using 109 different value selection heuristic configurations. Each of the following four tables summarizes the results for one set of problems; it contains 108 entries of the form $\langle A, B, C, D, E \rangle$, where:

- A is the acronym of the name of the heuristic. For example, PROBMOSTFAILS is denoted by **pmf**, and ADAPTIVEBESTSUCCESSRATIO by **absr**.
- B is the value of α that was used as the “initial score” of an unordered value.
- C is the value of either β or γ , depending on the heuristic: as described in Section 3, β is used to control the aggressiveness of LEASTFAILS, BESTSUCCESSRATIO, and their adaptive versions, while γ is used for all the rest.

D is the ratio of the average time that was required for the current heuristic configuration to find a single solution, over the time that was required for the RANDOM heuristic.

E is the ratio of the average normalized Hamming distance between the solutions that were found by the current heuristic configuration, over the distance that was achieved by the RANDOM heuristic.

All the entries in each table are sorted by the ratio $\frac{D}{E}$: the lower the ratio, the better is the performance/quality ratio relatively to the trade-off achieved by the RANDOM heuristic. *Each table is divided by a horizontal line into two parts: above the line appear the heuristic configurations with $D \leq E$ (and therefore we consider their trade-off to be better than that of RANDOM), and below the line are the heuristic configurations with $D > E$. (Note that RANDOM does not appear in the tables, since its $D = 1$ and $E = 1$.)*

As discussed at the beginning of Section 3, it is hard to achieve a better solution diversity than that of the RANDOM heuristic for such problems as those that are used in this benchmark. Indeed, in each table there are only a few entries with $E > 1$, and in all such entries the delta $E - 1$ is insignificant. On the other hand, there are many heuristic configurations whose solution diversity was much worse than that of RANDOM: for example, many configurations of LEASTFAILS, BESTSUCCESSRATIO, and their adaptive versions, with high aggressiveness and conservativeness, have low values of E .

The picture is different when comparing the *runtime* of the different heuristics to that of RANDOM. There are many heuristic configurations that run much faster than RANDOM; many non-probabilistic heuristics run twice as fast as RANDOM, or in some cases even 10 or 20 times faster (e.g., see the entries of LEASTFAILS in Table 1). Unfortunately (but expectedly), in many cases where the speed-up is very high, the loss of solution diversity is also very significant.

Nevertheless, at the beginning of each table there are at least several entries of heuristic configurations that achieve a relatively good speed-up without a significant loss of solution diversity. For example, at the beginning of Table 1 (set $\langle 50, 10, 225, 35 \rangle$) several configurations of BESTSUCCESSRATIO run about twice as fast as random ($0.498 \leq D \leq 0.593$) while losing only a relatively small portion of solution diversity ($0.916 \leq E \leq 0.929$). Some configurations of PROBLEASTFAILS, PROBBESTSUCCESSRATIO, and ADAPTIVEPROBBESTSUCCESSRATIO achieve a lesser speed-up ($0.798 \leq D \leq 0.838$), but almost do not lose any solution diversity ($0.951 \leq E \leq 0.991$).

The results in Table 2 are even better. All configurations of BESTSUCCESSRATIO achieve about $\times 4$ speed-up ($0.235 \leq D \leq 0.264$), while losing only about 15–20% of solution diversity ($0.761 \leq E \leq 0.843$). All configurations of PROBLEASTFAILS achieve lower speed-up ($0.701 \leq D \leq 0.802$), but their loss of solution diversity is insignificant ($0.955 \leq E \leq 0.982$). Many configurations of PROBBESTSUCCESSRATIO also achieve similar results. ADAPTIVEPROBLEASTFAILS is a little slower than the non-adaptive version, but it practically does not lose solution diversity.

Table 1. $(50, 10, 225, 35)$ results (relative to the RANDOM heuristic)

heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R
alf	0.25	0.05	0.062	0.12	apmf	2	0.25	0.937	0.98	pwsr	0.25	1	1.034	0.998
bsr	0.5	0.15	0.498	0.918	pbsr	0.5	0.25	0.942	0.985	pwsr	0.75	0.05	1.038	0.997
bsr	0.25	0.05	0.524	0.924	aplf	0.9	3	0.96	1.004	pbsr	0.25	0.05	1.029	0.985
bsr	0.5	0.45	0.531	0.92	apbsr	0.25	0.05	0.911	0.949	apwsr	0.75	0.5	0.973	0.926
bsr	0.25	0.45	0.546	0.921	aplf	0.5	0.25	0.966	1.002	alf	0.9	0.45	0.081	0.077
bsr	0.25	0.15	0.548	0.916	apwsr	0.5	0.5	0.9	0.931	pmf	2	0.25	1.033	0.976
lf	0.25	0.05	0.064	0.102	apwsr	0.25	0.05	0.916	0.947	apwsr	0.75	1	0.992	0.937
bsr	0.75	0.15	0.593	0.929	absr	0.25	0.45	0.972	0.998	pmf	1	0.25	1.042	0.984
bsr	0.75	0.45	0.626	0.923	apbsr	0.5	0.05	0.926	0.943	pmf	2	0.75	1.034	0.973
alf	0.25	0.15	0.094	0.136	alf	0.9	0.05	0.084	0.085	apbsr	1	0.25	1.005	0.932
bsr	0.75	0.05	0.655	0.911	absr	0.75	0.15	0.987	1.001	apbsr	0.5	0.25	1.039	0.96
lf	0.5	0.45	0.09	0.122	absr	0.5	0.05	0.988	1	pwsr	0.5	0.5	1.082	0.999
bsr	0.5	0.05	0.697	0.92	plf	0.9	3	0.974	0.984	lf	0.25	0.15	0.062	0.057
plf	0.9	0.75	0.826	0.991	plf	0.25	0.75	0.978	0.985	pmf	2	3	1.061	0.975
pbsr	1	0.25	0.824	0.986	plf	0.5	3	0.989	0.991	aplf	0.9	0.75	1.099	1
apbsr	1	0.05	0.798	0.951	apmf	0.5	3	0.982	0.98	apmf	2	3	1.082	0.98
pbsr	1	0.05	0.838	0.994	pwsr	0.75	0.5	1.004	1.001	alf	0.5	0.05	0.191	0.171
lf	0.5	0.15	0.05	0.058	absr	0.25	0.15	1.006	1.002	pmf	0.5	3	1.091	0.978
pwsr	0.25	0.5	0.868	1	plf	0.5	0.25	0.995	0.988	apwsr	0.5	1	1.051	0.933
plf	0.25	0.25	0.866	0.994	absr	0.75	0.05	1.008	1	alf	0.25	0.45	0.135	0.12
apmf	1	0.25	0.855	0.976	pwsr	0.75	1	1.01	1	alf	0.5	0.15	0.134	0.118
pbsr	0.5	0.05	0.88	0.986	apbsr	0.25	0.25	0.954	0.942	aplf	0.5	0.75	1.141	1.002
absr	0.5	0.45	0.899	1.001	pmf	0.5	0.25	0.99	0.978	apwsr	0.25	0.5	1.066	0.929
pbsr	1	0.5	0.886	0.982	aplf	0.25	0.25	1.015	1.002	apwsr	0.5	0.05	1.118	0.94
absr	0.5	0.15	0.904	1.001	absr	0.75	0.45	1.015	1.001	pmf	0.5	0.75	1.159	0.974
alf	0.5	0.45	0.11	0.121	pbsr	0.5	0.5	1.008	0.99	apmf	0.5	0.75	1.18	0.978
apmf	0.5	0.25	0.904	0.981	aplf	0.9	0.25	1.025	1.004	pmf	1	3	1.182	0.977
apbsr	1	0.5	0.867	0.938	pmf	1	0.75	0.997	0.974	apmf	1	0.75	1.186	0.977
aplf	0.5	3	0.928	1.001	plf	0.25	3	1.015	0.991	lf	0.25	0.45	0.099	0.082
apbsr	0.5	0.5	0.876	0.942	absr	0.25	0.05	1.023	0.998	alf	0.9	0.15	0.087	0.072
plf	0.9	0.25	0.932	0.991	pwsr	0.25	0.05	1.026	0.999	lf	0.9	0.05	0.081	0.064
aplf	0.25	0.75	0.945	1.002	pwsr	0.5	0.05	1.028	0.997	apwsr	0.25	1	1.183	0.928
apbsr	0.25	0.5	0.895	0.944	plf	0.5	0.75	1.019	0.988	apwsr	0.75	0.05	1.282	0.927
pbsr	0.25	0.25	0.936	0.985	aplf	0.25	3	1.034	1.002	lf	0.9	0.15	0.082	0.057
apmf	1	3	0.937	0.983	apmf	2	0.75	1.014	0.98	lf	0.9	0.45	0.13	0.085
pbsr	0.25	0.5	0.943	0.989	pwsr	0.5	1	1.034	0.999	lf	0.5	0.05	0.112	0.063

The effect on the ATGP benchmark is less prominent, but also noticeable. In Table 3 (ATGP $\langle 15, 21 \rangle$) the ADAPTIVEPROBLEASTFAILS with $\alpha = 0.25, \gamma = 0.25$ achieves about 17% speed-up while practically not losing any solution diversity ($D = 0.858, E = 0.991$), and several other heuristic configurations at the beginning of the table also achieve similar (although somewhat weaker) results. As expected, ATGP $\langle 26, 7 \rangle$ (Table 4) is less affected by the learning-reuse heuristics, since only a small number of solutions is required here. Still, ADAPTIVEPROBBESTSUCCESSRATIO with $\alpha = 1, \gamma = 0.05$ achieves about 12%

Table 2. $(50, 10, 225, 37)$ results (relative to the RANDOM heuristic)

heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R
bsr	0.5	0.05	0.239	0.835	lf	0.5	0.15	0.181	0.195	pwsr	0.75	0.05	1.251	0.985
bsr	0.5	0.15	0.245	0.829	absr	0.25	0.15	0.999	1.005	pwsr	0.5	0.5	1.286	1.006
bsr	0.75	0.15	0.245	0.82	absr	0.5	0.05	1.011	1	apmf	2	3	0.997	0.775
bsr	0.5	0.45	0.249	0.833	apbsr	0.5	0.5	0.868	0.858	apmf	2	0.75	1.073	0.822
bsr	0.75	0.05	0.254	0.843	absr	0.25	0.45	1.008	0.995	pwsr	0.75	1	1.304	0.985
bsr	0.25	0.15	0.24	0.786	apbsr	1	0.25	0.885	0.868	apwsr	0.75	0.5	1.12	0.846
bsr	0.75	0.45	0.25	0.808	absr	0.75	0.05	1.018	0.997	alf	0.9	0.45	0.235	0.176
bsr	0.25	0.45	0.235	0.761	absr	0.5	0.15	1.015	0.993	apmf	1	0.25	1.096	0.818
bsr	0.25	0.05	0.264	0.805	absr	0.75	0.45	1.037	1.005	apwsr	0.25	0.05	1.076	0.796
plf	0.5	0.75	0.701	0.962	absr	0.5	0.45	1.03	0.997	alf	0.5	0.05	0.172	0.124
plf	0.25	0.25	0.729	0.973	absr	0.75	0.15	1.04	1.006	apwsr	0.25	1	0.992	0.702
plf	0.9	0.75	0.737	0.955	absr	0.25	0.05	1.044	0.995	apmf	1	0.75	1.068	0.754
plf	0.9	0.25	0.754	0.969	apbsr	1	0.5	0.885	0.839	lf	0.5	0.05	0.254	0.178
pbsr	1	0.25	0.778	0.982	apbsr	0.25	0.25	0.913	0.862	apmf	0.5	0.75	1.124	0.786
plf	0.5	0.25	0.759	0.957	apbsr	0.25	0.5	0.904	0.847	alf	0.25	0.45	0.17	0.118
plf	0.9	3	0.76	0.958	apbsr	0.25	0.05	0.93	0.847	pmf	2	0.75	1.439	0.986
plf	0.25	3	0.76	0.957	apwsr	0.5	0.05	0.884	0.8	pmf	1	3	1.435	0.979
pbsr	0.25	0.25	0.781	0.983	apmf	2	0.25	0.997	0.901	alf	0.5	0.15	0.284	0.192
plf	0.5	3	0.769	0.96	apbsr	1	0.05	0.918	0.829	pmf	0.5	0.75	1.455	0.969
pbsr	0.5	0.25	0.775	0.957	apbsr	0.5	0.25	0.967	0.871	pmf	0.5	3	1.454	0.966
pbsr	1	0.05	0.807	0.994	apbsr	0.5	0.05	0.943	0.843	apwsr	0.75	0.05	1.154	0.758
pbsr	0.5	0.5	0.788	0.963	lf	0.25	0.15	0.204	0.182	pmf	2	3	1.511	0.989
plf	0.25	0.75	0.802	0.977	apwsr	0.5	1	0.924	0.813	lf	0.25	0.05	0.164	0.107
pbsr	1	0.5	0.804	0.979	lf	0.5	0.45	0.263	0.224	pmf	2	0.25	1.485	0.961
pbsr	0.5	0.05	0.818	0.993	apmf	1	3	1.013	0.848	alf	0.9	0.15	0.183	0.118
aplif	0.25	3	0.828	1.003	lf	0.25	0.45	0.179	0.15	pmf	0.5	0.25	1.459	0.937
pbsr	0.25	0.5	0.807	0.973	apmf	0.5	0.25	1.014	0.843	pmf	1	0.25	1.447	0.91
pbsr	0.25	0.05	0.815	0.981	apmf	0.5	3	1.031	0.841	apwsr	0.75	1	1.057	0.659
aplif	0.9	0.25	0.848	1.014	pwsr	0.5	0.05	1.254	1.018	alf	0.25	0.05	0.207	0.129
aplif	0.25	0.75	0.828	0.991	pwsr	0.75	0.5	1.234	0.998	pmf	1	0.75	1.473	0.911
aplif	0.9	0.75	0.837	0.999	pwsr	0.25	0.5	1.256	1.008	lf	0.9	0.45	0.272	0.167
aplif	0.5	3	0.842	1.001	pwsr	0.25	1	1.234	0.989	lf	0.9	0.15	0.219	0.123
aplif	0.9	3	0.841	0.986	pwsr	0.5	1	1.239	0.993	lf	0.9	0.05	0.222	0.124
aplif	0.25	0.25	0.859	0.996	pwsr	0.25	0.05	1.232	0.988	alf	0.25	0.15	0.263	0.125
aplif	0.5	0.75	0.85	0.975	apwsr	0.25	0.5	1.08	0.862	alf	0.5	0.45	0.282	0.132
aplif	0.5	0.25	0.878	0.972	apwsr	0.5	0.5	0.959	0.756	alf	0.9	0.05	0.271	0.095

speed-up while practically not losing any solution diversity, and some configurations of ADAPTIVEBESTSUCCESSRATIO and PROBBESTSUCCESSRATIO get similar (but a little weaker) results.

We can also categorize the heuristic groups according to their behavior. As expected, the heuristics LEASTFAILS and BESTSUCCESSRATIO, which attempt to find a solution as fast as possible, usually achieve very high speed-up, but it is often accompanied by a significant loss of solution diversity; their adaptive versions also show similar behavior (although less dramatically). Nevertheless,

Table 3. ATGP $\langle 15, 21 \rangle$ benchmark results (relative to the RANDOM heuristic)

heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R
aplf	0.25	0.25	0.858	0.991	aplf	0.25	0.75	0.996	0.993	apwsr	0.75	1	1.133	0.864
plf	0.5	0.75	0.879	0.985	apbsr	0.5	0.25	0.925	0.92	pmf	1	0.75	1.428	0.995
bsr	0.75	0.45	0.874	0.975	apbsr	0.5	0.5	0.9	0.895	apwsr	0.25	1	1.101	0.76
aplf	0.5	0.25	0.901	0.985	apmf	2	0.75	0.977	0.97	pwsr	0.25	0.05	1.555	0.997
absr	0.5	0.05	0.809	0.874	absr	0.75	0.45	0.997	0.989	pwsr	0.75	0.05	1.594	1.004
pbsr	0.5	0.05	0.898	0.965	aplf	0.9	0.75	1.008	0.989	pwsr	0.5	0.05	1.669	1.012
pbsr	0.5	0.5	0.921	0.987	plf	0.25	0.75	0.983	0.964	pmf	0.5	3	2.04	0.95
plf	0.9	3	0.936	1	apwsr	0.75	0.05	1.017	0.996	pmf	2	0.25	3.191	0.977
apwsr	0.5	0.05	0.918	0.976	pwsr	0.75	1	1.031	1.006	pmf	1	0.25	3.76	0.975
pbsr	0.25	0.05	0.889	0.944	aplf	0.9	0.25	1.011	0.983	pmf	0.5	0.25	3.751	0.961
absr	0.75	0.05	0.894	0.946	aplf	0.25	3	1.025	0.989	bsr	0.25	0.45	0.536	0.035
plf	0.25	3	0.944	0.997	pbsr	0.25	0.5	1.009	0.973	bsr	0.25	0.15	0.623	0.033
plf	0.9	0.25	0.915	0.966	pmf	1	3	1.024	0.978	bsr	0.25	0.05	0.624	0.033
apmf	2	3	0.926	0.972	apmf	1	0.75	1.024	0.968	alf	0.9	0.45	0.536	0.025
pbsr	0.25	0.25	0.932	0.977	plf	0.25	0.25	0.965	0.906	alf	0.5	0.45	0.538	0.025
aplf	0.5	0.75	0.949	0.995	pmf	2	3	1.063	0.997	alf	0.9	0.05	0.539	0.025
aplf	0.9	3	0.966	1.007	plf	0.5	0.25	0.991	0.926	alf	0.9	0.15	0.539	0.025
pbsr	1	0.05	0.935	0.97	pwsr	0.25	0.5	1.071	1.001	lf	0.9	0.15	0.541	0.025
apbsr	1	0.25	0.924	0.957	apwsr	0.75	0.5	0.984	0.914	bsr	0.5	0.45	0.536	0.025
pbsr	1	0.5	0.951	0.984	apbsr	0.25	0.25	0.948	0.881	lf	0.9	0.45	0.543	0.025
apbsr	1	0.5	0.91	0.935	apmf	0.5	0.25	1.048	0.963	alf	0.2	0.45	0.526	0.018
pbsr	1	0.25	0.956	0.978	apmf	1	3	1.038	0.954	alf	0.2	0.15	0.526	0.018
apbsr	0.5	0.05	0.948	0.969	pwsr	0.75	0.5	1.092	1.001	alf	0.5	0.05	0.527	0.018
pbsr	0.5	0.25	0.956	0.976	apbsr	0.25	0.5	0.96	0.872	alf	0.5	0.15	0.527	0.018
apmf	2	0.25	0.975	0.994	apmf	0.5	0.75	1.053	0.952	alf	0.2	0.05	0.528	0.018
aplf	0.5	3	0.993	1.009	absr	0.25	0.05	0.834	0.742	lf	0.9	0.05	0.529	0.018
apbsr	1	0.05	0.964	0.979	pwsr	0.5	0.5	1.121	0.99	bsr	0.75	0.15	0.529	0.018
apbsr	0.25	0.05	0.947	0.959	apmf	1	0.25	1.088	0.956	lf	0.2	0.05	0.529	0.018
absr	0.75	0.15	0.979	0.988	apwsr	0.5	0.5	1.029	0.892	lf	0.5	0.05	0.529	0.018
plf	0.9	0.75	0.967	0.973	pwsr	0.25	1	1.142	0.983	bsr	0.75	0.05	0.53	0.018
absr	0.5	0.15	0.981	0.986	apmf	0.5	3	1.111	0.921	lf	0.2	0.45	0.53	0.018
absr	0.25	0.45	0.983	0.988	pmf	0.5	0.75	1.198	0.977	lf	0.2	0.15	0.531	0.018
plf	0.5	3	0.99	0.993	apwsr	0.5	1	1.009	0.818	lf	0.5	0.45	0.533	0.018
absr	0.5	0.45	0.991	0.993	apwsr	0.25	0.5	1.037	0.827	lf	0.5	0.15	0.533	0.018
pwsr	0.5	1	0.992	0.992	apwsr	0.25	0.05	1.232	0.974	bsr	0.5	0.15	0.526	0.018
absr	0.25	0.15	0.933	0.93	pmf	2	0.75	1.307	1.008	bsr	0.5	0.05	0.526	0.018

in the randomly generated sets the solution diversity loss by these heuristics was often much lower than the speed-up gain. Note also that such heuristics can achieve good solution diversity when lowering the aggressiveness and conservativeness considerably: for example, BESTSUCCESSRATIO with $\alpha = 0.75$, $\beta = 0.45$ gets $E = 0.975$ for ATGP $\langle 15, 21 \rangle$ (Table 3) and high values for other problem sets as well, while its runtime is relatively low.

The heuristics PROBMOSTFAILS and PROBWORSTSUCCESSRATIO and their adaptive versions usually do not achieve a better solution diversity than RANDOM, but can be much slower.

Table 4. ATGP(26, 7) benchmark results (relative to the RANDOM heuristic)

heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R	heur	α	$\beta \gamma$	time/R	HD/R
apbsr	1	0.05	0.89	0.989	apbsr	0.5	0.05	0.904	0.821	absr	0.5	0.05	0.773	0.52
absr	0.75	0.15	0.9	0.978	apbsr	0.25	0.05	0.872	0.792	apmf	1	0.75	1.384	0.876
pbsr	1	0.05	0.912	0.984	pmf	2	0.25	1.085	0.981	apwsr	0.75	1	0.987	0.582
absr	0.5	0.45	0.929	0.994	pwsr	0.5	0.5	1.109	1.002	apmf	1	3	1.439	0.82
absr	0.75	0.45	0.932	0.994	apwsr	0.5	0.05	0.991	0.894	apwsr	0.75	0.5	1.251	0.654
bsr	0.75	0.45	0.899	0.957	pwsr	0.25	0.05	1.099	0.986	apwsr	0.5	0.5	1.141	0.594
aplf	0.9	0.75	0.95	0.997	pbsr	0.5	0.25	0.936	0.837	absr	0.25	0.05	0.736	0.37
aplf	0.9	3	1.01	1.059	pmf	0.5	3	0.949	0.844	apwsr	0.5	1	1.279	0.525
plf	0.25	3	0.904	0.942	pbsr	0.25	0.05	0.909	0.792	apwsr	0.25	0.5	1.353	0.527
aplf	0.5	0.25	0.887	0.923	pwsr	0.5	1	1.033	0.897	apwsr	0.25	1	1.26	0.468
absr	0.5	0.15	0.838	0.863	apbsr	0.5	0.5	0.87	0.754	alf	0.2	0.45	0.673	0.179
absr	0.25	0.45	0.96	0.987	apbsr	1	0.25	0.948	0.817	lf	0.5	0.05	0.674	0.179
aplf	0.25	3	0.958	0.982	pmf	1	3	0.972	0.833	lf	0.5	0.45	0.675	0.179
pbsr	1	0.25	0.983	1.003	apmf	0.5	0.25	1.11	0.94	lf	0.2	0.05	0.675	0.179
plf	0.9	3	0.951	0.958	apbsr	0.25	0.25	0.907	0.755	lf	0.2	0.15	0.676	0.179
aplf	0.9	0.25	0.936	0.941	plf	0.9	0.25	1.016	0.844	lf	0.2	0.45	0.677	0.179
pbsr	0.25	0.25	0.941	0.943	apmf	2	0.25	1.173	0.975	lf	0.5	0.15	0.677	0.179
aplf	0.5	0.75	1.048	1.047	pwsr	0.25	1	1.007	0.831	lf	0.9	0.05	0.678	0.179
aplf	0.5	3	0.981	0.973	pmf	0.5	0.75	1.156	0.95	alf	0.2	0.05	0.681	0.179
pwsr	0.5	0.05	1.002	0.991	plf	0.5	0.75	0.987	0.81	alf	0.5	0.05	0.682	0.179
aplf	0.25	0.75	1.026	1.004	pwsr	0.25	0.5	1.025	0.839	alf	0.2	0.15	0.693	0.179
pmf	1	0.25	1.032	1	apmf	2	3	1.13	0.911	alf	0.5	0.15	0.709	0.179
pmf	2	3	0.996	0.963	absr	0.75	0.05	0.787	0.632	bsr	0.5	0.45	0.679	0.164
pbsr	1	0.5	0.961	0.92	apbsr	1	0.5	1.049	0.831	bsr	0.75	0.05	0.677	0.158
pmf	2	0.75	0.996	0.944	apmf	2	0.75	1.154	0.912	bsr	0.75	0.15	0.692	0.158
pwsr	0.75	0.05	1.085	1.02	apbsr	0.5	0.25	0.976	0.766	bsr	0.5	0.05	0.682	0.155
pwsr	0.75	1	1.021	0.954	pbsr	0.25	0.5	1.001	0.775	bsr	0.25	0.05	0.707	0.155
pmf	1	0.75	1.05	0.981	apmf	0.5	0.75	1.173	0.907	bsr	0.5	0.15	0.708	0.155
pwsr	0.75	0.5	1.037	0.969	apwsr	0.25	0.05	1.057	0.801	alf	0.5	0.45	0.668	0.144
pbsr	0.5	0.05	0.927	0.863	plf	0.5	0.25	1.119	0.829	bsr	0.25	0.15	0.7	0.149
plf	0.25	0.75	0.926	0.859	apmf	0.5	3	1.078	0.785	alf	0.9	0.05	0.694	0.144
plf	0.9	0.75	0.995	0.922	apmf	1	0.25	1.337	0.959	lf	0.9	0.45	0.702	0.142
pbsr	0.5	0.5	0.94	0.869	apbsr	0.25	0.5	0.899	0.637	lf	0.9	0.15	0.706	0.142
plf	0.5	3	1.025	0.947	absr	0.25	0.15	0.819	0.579	alf	0.9	0.45	0.71	0.142
pmf	0.5	0.25	1.008	0.923	apwsr	0.75	0.05	1.224	0.845	alf	0.9	0.15	0.711	0.142
aplf	0.25	0.25	1.013	0.927	plf	0.25	0.25	0.913	0.624	bsr	0.25	0.45	0.669	0.131

On the other hand, the PROBLEASTFAILS, PROBBESTSUCCESSRATIO, and their adaptive versions often achieve a moderate speed-up without sacrificing much solution diversity, as can be seen in all the parts of our benchmark. The adaptive versions usually achieve somewhat higher solution diversity, but are slower than the non-adaptive versions.

Note also that in all the above results the performance was measured as a function of runtime. When measuring other parameters, such as number of failures or the number of the search-tree nodes that were traversed during the search, the relative gain of all the above learning-reuse heuristics comparing

to the RANDOM heuristic is much higher. This is easily explained by the fact that the value selection function of RANDOM is much faster than that of its competitors, and by the fact that the usage of RANDOM incurs no overhead of the learning functionality, which is required by the learning-reuse heuristics.

5 Summary and Further Research

We have analyzed the trade-off between the CSP search performance and the solution diversity, as a function of the value-ordering heuristic. We have also suggested methods to categorize and control this trade-off using different heuristics and their parameters. The heuristics were tested on a benchmark of randomly generated CSPs and on real-life ATG problems, where the goal was to find many diverse solutions as fast as possible. The results show that while the simple RANDOM heuristic reaches quite reasonable solution diversity, there are learning-reuse heuristics that can reach almost the same level of diversity with a significant performance speed-up.

We have also suggested using adaptive heuristic versions that modify the value selection parameters depending on the attributes of each variable. While in many cases this modification has improved the performance/diversity trade-off of the heuristic, the implementation of the adaptive heuristic versions in our experiment did not reach its full potential due to the difficulty of efficiently identifying relatively independent subsets of variables, as discussed in Section 3.1. Continuing the research in this direction can probably improve the effectiveness of the adaptive heuristics; we leave this for further research.

It is also worth mentioning other interesting related directions for future research, such as formal analysis of the relation between the solution distribution and solution diversity, influence of variable selection heuristics on solution diversity, etc.

Acknowledgments. The author is very grateful to Anna Moss and Boris Gutkovich for valuable comments and discussions, and to Alexander Libov for a very useful framework of gathering statistics of the ATGP experiments. The author is also grateful to the anonymous referees for many useful suggestions.

References

1. Dechter, R., Kask, K., Bin, E., Emek, R.: Generating random solutions for constraint satisfaction problems. In: AAI 2002, pp. 15–21 (2002)
2. Geelen, P.A.: Dual viewpoint heuristics for binary constraint satisfaction problems. In: 10th Europ. Conf. on Artif. Intell. (ECAI 1992), pp. 31–35 (1992)
3. Gogate, V., Dechter, R.: A new algorithm for sampling CSP solutions uniformly at random. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 711–715. Springer, Heidelberg (2006)
4. Gutkovich, B., Moss, A.: CP with architectural state lookup for functional test generation. In: 11th Annu. IEEE Internat. Worksh. on High Level Design Validation and Test, pp. 111–118 (2006)

5. Hebrard, E., Hnich, B., O'Sullivan, B., Walsh, T.: Finding diverse and similar solutions in constraint programming. In: AAAI 2005, pp. 372–377 (2005)
6. Van Hentenryck, P., Coffrin, C., Gutkovich, B.: Constraint-based local search for the automatic generation of architectural tests. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 787–801. Springer, Heidelberg (2009)
7. Intel® 64 and IA-32 architectures software developer's manual (2009)
8. Koller, D., Megiddo, N.: Constructing small sample spaces satisfying given constraints. *SIAM J. Disc. Math.* 7(2), 260–274 (1994)
9. Larkin, D.: Generating random solutions from a constraint satisfaction problem with controlled probability. In: 1st Internat. Worksh. on Constraints in Functional Verification of CP 2002 (2002)
10. Mittal, S., Falkenhainer, B.: Dynamic constraint satisfaction problems. In: 8th National Conf. on Artif. Intell., pp. 25–32 (1990)
11. Moss, A.: Constraint patterns and search procedures for CP-based random test generation. In: Yorav, K. (ed.) HVC 2007. LNCS, vol. 4899, pp. 86–103. Springer, Heidelberg (2008)
12. Moss, A., Gutkovich, B.: Functional test generation with distribution constraints. In: 5th Internat. Haifa Verification Conf. (HVC 2009). LNCS (to appear, 2010)
13. Naveh, Y., Rimon, M., Jaeger, I., Katz, Y., Vinov, M., Marcus, E., Shurek, G.: Constraint-based random stimuli generation for hardware verification. *AI Magazine* 28(3), 13–30 (2007)
14. Refalo, P.: Impact-based search strategies for constraint programming. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 557–571. Springer, Heidelberg (2004)
15. Schreiber, Y.: Cost-driven interactive CSP with constraint relaxation. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 707–722. Springer, Heidelberg (2009)
16. Smith, B.M.: Phase transition and the mushy region in constraint satisfaction problems. In: ECAI 1994, pp. 100–104 (1994)
17. Zhang, Z., Epstein, S.L.: Learned value-ordering heuristics for constraint satisfaction. In: STAIR 2008 Workshop at AAAI (2008)

A New $\mathcal{O}(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints*

Andreas Schutt¹ and Armin Wolf²

¹ National ICT Australia, Department of Computer Science & Software Engineering,
The University of Melbourne, Australia
`aschutt@csse.unimelb.edu.au`

² Fraunhofer FIRST, Kekuléstr. 7, D-12489 Berlin, Germany
`armin.wolf@first.fraunhofer.de`

Abstract. The recent success of the lazy clause generator (a hybrid of a FD and a SAT solver) on resource-constrained project scheduling problems (RCPSP) shows the importance of the global cumulative constraint to tackle these problems. A key for an efficient cumulative propagator is a fast and correct pruning of time-bounds. The not-first/not-last rule (which is not subsumed by other rules) detects activities that cannot be run at first/last regarding to an activity set and prunes their time bounds. This paper presents a new sound not-first/not-last pruning algorithm which runs in $\mathcal{O}(n^2 \log n)$, where n is the number of activities. It may not find the best adjustments in the first run, but after at most n iterations. This approach of iteration fits the setup of constraint propagation quite naturally offering the opportunity that a fixed point is reached more efficiently. Moreover, it uses a novel approach of generation of some “artificial” activities in the context of triggering pruning rules correctly. In experiments on RCPSP amongst others from the well-established PSPLib we show that the algorithm runs negligible more often than a complete algorithm while taking its advantage from the lower – to the best of our knowledge the lowest known – runtime complexity.

1 Introduction

Scheduling of capacity-consuming activities on commonly shared cumulative resources has strong practical relevance. There, the activities have to be scheduled such that the capacities of the cumulative resources are never exceeded. Such problems occur in resource-constrained project scheduling (RCPSP) as well as in production planning if resources like water, fuel, electric power, consumables or even human skills have to be assigned to activities requiring these resources. Due to its practical relevance and its high complexity – these problems are in general NP-complete in the strong sense [4] – scheduling on cumulative resources is one important research topic in constraint-based scheduling (c.f. [2,3,5,9]). Thanks

* This work was partly funded by the European Union (EFRE) and the state Berlin, grant no. 10023515 and by the German Federal Ministry of Education and Research (BMBF), grant no. 13N10598. The core of the published results is already presented in the master thesis of Schutt [11].

to the “energetic nature” of cumulative scheduling the achieved results are also valid for other scheduling problems like scheduling of activities on alternative disjunctive resources or non-overlapping placement of rectangles in a restricted two-dimensional area (for some recent applications see e.g. [15,19]).

Generalized pruning rules exist that are adapted from disjunctive resource scheduling as well as algorithms performing these generalized rules – *overload checking*, *(extended) edge-finding*, and *not-first/not-last detection*. However, a detailed examination of some of these algorithms presented in [2,9] showed that they are incomplete, i.e. pruning, which is possible regarding the rules, is not performed by the corresponding algorithms, resulting in weak search space reductions. Mercier and van Hentenryck [8] and ourselves independently recognized the incompleteness of (extended) edge-finding and, in parallel, worked out complete algorithms that have the same runtime complexity (not presented here).

In [14] we showed the incorrectness as well as the incompleteness of the not-first/not-last pruning algorithms presented in [9]. Furthermore, we presented a sophisticated and complete not-first/not-last pruning algorithm reducing the $\mathcal{O}(n^4)$ time complexity of a naive algorithm to $\mathcal{O}(n^3 \log n)$ where n is the number of the considered activities. This algorithm uses balanced binary trees similar to the algorithms for disjunctive resource scheduling presented in [16,18]. To our knowledge, our $\mathcal{O}(n^3 \log n)$ algorithm is currently the best known (complete) algorithm for *not-first/not-last detection* according to runtime complexity.

Recently, Schutt et al. [13,12] used the lazy clause generator [10] (a hybrid of a finite domain and a SAT solver) to explain the propagation of the cumulative constraint. They take the advantage of the efficient no-good learning facility and the conflict-driven search from the SAT solver in order to minimize the makespan of RCPSP. Currently, this method is the best exact method known so far.

In this paper we will present a new $\mathcal{O}(n^2 \log n)$ algorithm which may perform weaker pruning per iteration based on *not-first/not-last detection*, but has the same pruning power as a complete algorithm if it is iterated sufficiently often. In order to achieve the desired propagation “artificial” activities, i.e. *pseudo-tasks*, are introduced what is to our best knowledge a new approach (in this context). Experiments on RCPSP instances taken from [11,7] indicate that the new algorithm only needs a few iterations for its completeness, and it is faster than the other published not-first/not-fast algorithms especially on large instances.

The paper is organized as follows. In the next section the considered *cumulative scheduling problems* are formally defined and then the *not-last detection* is presented on the more general pruning rule already introduced in [14,9]. Following this, the new not-last algorithm is presented and its correctness and reduced complexity is formally proven. The same section introduces balanced binary trees which are used for efficiency reasons by this algorithm. The paper concludes with the results of some experiments, a summary of the performed and future work.

2 Notations

The (*non-preemptive*) *cumulative scheduling problem* (CuSP) represents the fact that some activities or *tasks* i, j, t, \dots requiring capacities c_i, c_j, c_t, \dots have to be

¹ Considerations about not-first detection are omitted due to the duality of both rules.

scheduled on a common resource offering a constant capacity C such that the tasks are processed without interruption and without exceeding the capacity of the resource at any time. Formally, this problem is defined as follows:

Definition 1 (Cumulative Scheduling Problem – adopted from [8]). A cumulative scheduling problem (*CuSP*) is specified by a cumulative resource of capacity C and a finite task set T . Each task $i \in T$ is specified by its release date r_i , its due date or deadline d_i , its processing time p_i , and its capacity requirement c_i , all being natural numbers.

The problem is to find a schedule, i.e. a solution that assigns a start time s_i to each task $i \in T$ such that this start time and its end time $s_i + p_i$ are feasible (left expression), and the cumulative constraint (right expression) is satisfied for all time units τ :

$$\forall i \in T : r_i \leq s_i \leq s_i + p_i \leq d_i, \quad \forall \tau : \sum_{i \in T, s_i \leq \tau < s_i + p_i} c_i \leq C$$

The *CuSP* is solvable if such a schedule exists; otherwise it is unsolvable. Finally, let $n := |T|$ be the number of tasks and $e_i := c_i \cdot p_i$ be the energy of any task $i \in T$.

In the following, we assume an underlying *CuSP* with its resource and tasks as specified in Definition 1, unless otherwise stated. We also generalize the notions of release dates, due dates or deadlines, and energies for task sets, i.e.

$$r_\Omega := \min_{j \in \Omega} r_j, \quad d_\Omega := \max_{j \in \Omega} d_j, \quad \text{and} \quad e_\Omega := \sum_{j \in \Omega} e_j$$

where $\Omega \subseteq T$ is any task set, especially the empty set. In this special case let $r_\emptyset := -\infty$, $d_\emptyset := \infty$, and $e_\emptyset := 0$.

A constraint-based scheduling approach to *CuSP* checks in general necessary conditions for the existence of a solution and further uses pruning rules to get rid of infeasible start times or end times that are not part of any solution. Obviously, a *CuSP* has no schedule if there is a subset of tasks which requires within its release date and its deadline more energy than available. This leads directly to the following definition:

Definition 2 (E-Feasibility). A *CuSP* specified by a task set T and a capacity C is E-feasible if $\forall \Omega \subseteq T : C \cdot (d_\Omega - r_\Omega) \geq e_\Omega$ holds.

In the following, we assume that the underlying *CuSP* is E-feasible, unless otherwise stated. E-feasibility can be checked with $\mathcal{O}(n \log n)$ time complexity (cf. [19]).

2.1 The Not-Last Detection Rule

In addition to the edge-finding rules, not-first/not-last detection rules offer another fundamental technique to remove infeasible start and end times in disjunctive as well as in cumulative scheduling. Due to the symmetry of these rules, the work presented in this paper focuses on the pruning of the tasks' end times while using the not-last detection rule.²

² The adaptation for not-first detection is straightforward and thus omitted.

The not-last detection rule is the pendant to the edge-finding instance that finds the “edge” between a task that ends after some other tasks end – the “edge” is the latest task. Not-last detection finds a task that is “not-last”: it must end before at least one of some other tasks starts, i.e. the detected task is not the latest. In other words, there is no feasible schedule of the underlying CuSP such that the end time of the detected task is greater than the *latest start times* of the other tasks.

Definition 3 (Latest Start Time). *Let a CuSP be specified by a task set T and a capacity C . The latest start time of a task $i \in T$ is defined as $lst_i := d_i - p_i$. The latest start time of a task set Ω is defined as $lstmax_\Omega := \max\{lst_i \mid i \in \Omega\}$ for $\Omega \neq \emptyset$ and $lstmax_\emptyset := -\infty$.*

With these notions, the core of the not-last detection rule is formally given by the following proposition:

Proposition 1 (The (Simplified) Not-Last Detection Rule). *Let a CuSP be specified by a task set T and a capacity C . Then, for each subset of tasks $\Omega \subseteq T$ and each task $i \in T \setminus \Omega$ it holds: If the not-last condition*

$$lstmax_\Omega < d_i \wedge e_\Omega + c_i(d_\Omega - \max(lst_i, r_\Omega)) > C(d_\Omega - r_\Omega) \tag{1}$$

is satisfied, then all end times later than $lstmax_\Omega$ are inconsistent, i.e. for each schedule of the CuSP it holds that $s_i + p_i \leq lstmax_\Omega$.

Proof. The proof is shown in [14]. □

Definition 4 (Least Upper Bound). *The least upper bound $UB[i]$ on the due date of a task $i \in T$ regarding the rule is defined as*

$$UB[i] := \min_{\Omega \subseteq T \setminus \{i\} | \alpha} lstmax_\Omega$$

where α holds if the condition (1) is satisfied for i and Ω .

The following example shows the application of the rule and is used as running example throughout the rest of the paper.

Example 1. Let $(T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{i}\}, C = 2)$ be CuSP instance with

	r	d	c	p	e
\mathbf{a}	5	9	2	2	4
\mathbf{b}	7	10	1	1	1
\mathbf{c}	9	10	1	1	1
\mathbf{i}	1	10	1	6	6

The not-last detection rule only holds for \mathbf{i} and $\Omega = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ because $lstmax_\Omega = 9 < 10 = d_i$ and $e_\Omega + c_i(d_\Omega - \max(lst_i, r_\Omega)) = 6 + 1 \cdot (10 - 5) = 11 > 10 = C(d_\Omega - r_\Omega)$. Hence, $UB[\mathbf{i}] = 9$.

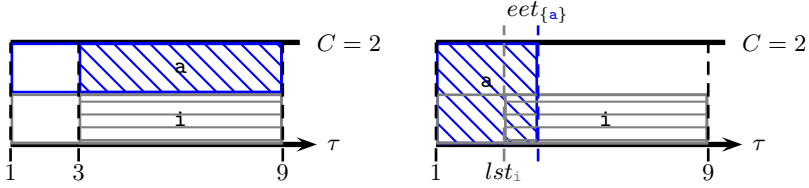


Fig. 1. An example that Vilím’s algorithm is not applicable for CuSP. **Left:** The tasks *a* and *i* can be run in parallel. **Right:** The earliest end time $eet_{\{a\}}$ is greater than lst_i , Vilím’s algorithm would incorrectly restrict d_i to $lst_a = 3$.

A complete algorithm regarding the not-last rule finds the least upper bound $UB[i]$ for any task i . An incomplete algorithm which will be introduced in the next section finds an upper bound $UB'[i]$ such that $UB[i] \leq UB'[i] < d_i$ holds if $UB[i] < d_i$ and $UB'[i] = d_i$ otherwise. This means that a weaker algorithm must be run several times to find the least upper bound for all tasks.

In the following considerations about an efficient algorithm applying the not-last rule the *earliest end time* of a task set plays an important role. This is a non-trivial lower bound of the end times of all tasks in a task set where different release dates of the tasks are taken into account:

Definition 5 (Earliest End Time). Let a CuSP be specified by a task set T and a capacity C . The earliest start time of a task $i \in T$ is defined as $eet_i := r_i + p_i$. The earliest end time of a task set $\Omega \subseteq T$ is defined as

$$eet_{\Omega} := \max\{r_{\Omega'} + \frac{e_{\Omega'}}{C} \mid \Omega' \subseteq \Omega\} .$$

3 A New Not-Last Algorithm

In this section we introduce a new not-last algorithm with $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n)$ space complexity. It is based on Vilím’s not-last algorithm for the **disjunctive** case of the CuSP where $C = 1$ holds. Vilím’s not-last algorithm [16] works essentially as follows: For each task $i \in T$ it checks whether there is a task $j \in T$ such that $eet_{\Omega \setminus \{i\}} > lst_i$ holds for the task set $\Omega = \{k \in T \mid lst_k \leq lst_j < d_i\}$. If so, it updates $d_i = lst_j$. This is correct in the **disjunctive** case, but unfortunately it cannot be applied in the general case of the CuSP with $C > 1$ as the following example shows.

Example 2. Let $T = \{a, i\}$ and $C = 2$ specify a CuSP with $r_a = r_i = 1$, $d_a = d_i = 9$, $p_a = p_i = 6$, $c_a = c_i = 1$ and $e_a = e_i = 6$. Considering the task *i* there is the task *a* with $lst_a < d_i$ and $eet_{\Omega \setminus \{i\}} = eet_{\{a\}} = 7$. Due to the fact that $lst_a = lst_i = 3$ holds, Vilím’s algorithm restricts the latest end time d_i wrongly to 3 because the not-last condition is not satisfied (see Fig. 1).

Nevertheless, at first we describe how the earliest end times of task sets can be used to check the not-last condition in spite of the approach followed in

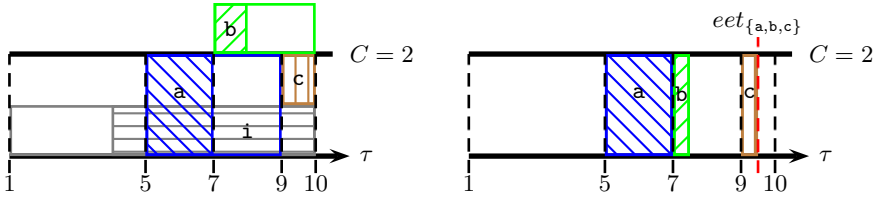


Fig. 2. The instance from Ex. 3 is shown. **Left:** The normal view of the tasks. **Right:** The energetic view of the tasks for calculation of $eet_{\{a,b,c\}} = 9.5$.

Vilím’s algorithm. At second, the usage of a binary balanced tree for an efficient computation of the earliest end times of task sets is described and a new not-last algorithm is presented. At last the correctness of the computation based on a balanced tree and of the new algorithm is proven as well as its complexity.

A task $i \in T$ and a task set $\Omega \subseteq T \setminus \{i\}$ satisfy the not-last condition if $lst_{max\Omega} < d_i$ and $e_\Omega + c_i(d_\Omega - \max(lst_i, r_\Omega)) > C(d_\Omega - r_\Omega)$ hold. Thus adapting Vilím’s approach directly for the CuSP in general the following inequality has to be checked for the earliest end time of any task set $\Omega \subseteq T \setminus \{i\}$:

$$eet_\Omega > d_{\Omega'} - \frac{c_i}{C}(d_{\Omega'} - \max(lst_i, r_{\Omega'})) \tag{2}$$

where $\Omega' \subseteq \Omega$ satisfies $eet_\Omega = r_{\Omega'} + e_{\Omega'}/C$. If this inequality holds then the not-last rule applies for i and Ω' , but the converse argument does not hold in general. This means that if the inequality does not apply for Ω then there can still exist a subset of Ω that satisfies the not-last rule. At next an example is given that demonstrates this issue and shows that is not sufficient to only consider $\Omega(i) := \{j \in T \setminus \{i\} \mid lst_j < d_i\}$ as for the disjunctive case.

Example 3. Let us consider our example from Ex. 1 and what is shown in Fig. 2. The not-last rule only holds for the task i and the task set $\Omega(i) = \{a, b, c\}$. However, since its subset $\Omega' = \{c\}$ determines the earliest end time $eet_{\{a,b,c\}} = 9.5$ the inequality (2) is not satisfied.

A closer analysis of Ex. 3 shows that inequality (2) is not satisfied since the required energy for i ’s execution is not considered if i starts at its latest start time. Therefore, our idea is to model this energy $c_i(d_{\Omega(i)} - lst_i)$ that is not free for the tasks in $\Omega(i)$ as “artificial” tasks. We call them *pseudo-tasks* in order to distinguish them from tasks in T and denote their set as S . Thus, the energy which is not free for the tasks in $\Omega(i)$ is the sum of the energy $c_i(\min(d_{\Omega(i)}, d_i) - lst_i)$ which is reserved for i ’s execution, and $c_i(d_{\Omega(i)} - \min(d_{\Omega(i)}, d_i))$ which is blocked after i ’s end because each task $j \in \Omega(i)$ starts by definition before d_i .

Definition 6 (Pseudo-Tasks). Let a task set T be given. Further let $Z = z_1, z_2, \dots, z_m$ be the sequence of the release and due dates in $\{r_j \mid j \in T\} \cup \{d_j \mid j \in T\}$ sorted in increasing order. For a task $i \in T$ let l be the smallest index with $lst_i < z_l$. Then $S(i) = \{s_1, \dots, s_{m-l+1}\}$ is the set of pseudo-tasks that is composed as follows where $f(k) = k - l + 1$ for $k = l, \dots, m$

$$r_{s_{f(k)}} = z_{k-1}, \quad d_{s_{f(k)}} = z_k, \quad c_{s_{f(k)}} = c_i, \quad p_{s_{f(k)}} = d_{s_{f(k)}} - r_{s_{f(k)}} .$$

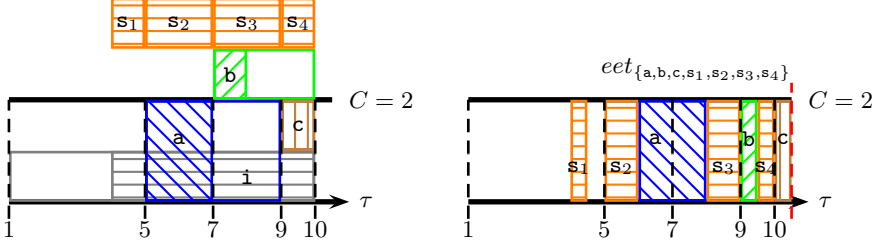


Fig. 3. The figure depicts the instance from Ex. 3 with the corresponding pseudo-tasks s_1, s_2, s_3, s_4 concerning i where i and $\{a, b, c\}$ satisfy the not-last rule. **Left:** The normal view of all tasks. **Right:** The energetic view of all tasks for the determination of $eet_{\{a,b,c,s_1,s_2,s_3,s_4\}} = 10.5$.

Note that for $n = |T|$ at most $(2n - 1)$ pseudo-tasks are generated for any task $j \in T$ because at most $2n$ different release and due dates exist in Z . In Fig. 3 the pseudo-tasks for task i are shown for the CuSP in Ex. 3.

The following theorem shows how pseudo-tasks determine the tasks sets that satisfy the condition of the not-last rule:

Theorem 1. *Let a CuSP be specified by a task set T and a capacity C . For each task $i \in T$ there exists a task set $\Omega \subseteq T \setminus \{i\}$ satisfying the condition of the not-last rule if and only if there is a task j in $\Omega(i) = \{k \in T \setminus \{i\} \mid lst_k < d_i\}$ such that for $S(i, d_j) = \{s \in S(i) \mid d_s \leq d_j\}$ holds*

$$eet_{\Omega(i,d_j) \cup S(i,d_j)} > d_j . \tag{3}$$

The proof of Theorem 1 uses the following lemma which shows: if a task set $\Psi' \subseteq \Psi$ determines eet_Ψ then it includes all tasks $t \in \Psi$ with $r_{\Psi'} \leq r_t$.

Lemma 1. *Let Ψ be a set of tasks, C be a resource capacity, and $\Psi' \subseteq \Psi$ with $r_{\Psi'} + e_{\Psi'}/C = eet_\Psi$. Then it holds: $\Psi' = \{t \in \Psi \mid r_{\Psi'} \leq r_t\}$.*

Proof (by contradiction). Let us assume that there is a task $t \in \Psi$ with $r_{\Psi'} \leq r_t$ and $t \notin \Psi'$. Because of $e_t > 0$ it holds $eet_\Psi = r_{\Psi'} + e_{\Psi'}/C < r_{\Psi' \cup \{t\}} + e_{\Psi' \cup \{t\}}/C$, which contradicts the choice of Ψ' . \square

Proof (of Theorem 1). Let CuSP be specified by a task set T and a capacity C and $i \in T$ be any task. At first we show the forward and then the backward direction of the equivalence.

“ \Rightarrow ”: There is a subset $\Omega \subseteq T \setminus \{i\}$ such that the condition of the not-last rule holds for Ω and i , i.e. $lstmax_\Omega < d_i$ and $e_\Omega + c_i(d_\Omega - \max(lst_i, r_\Omega)) > C(d_\Omega - r_\Omega)$. We have to show the existence of a task $j \in \Omega(i)$ that satisfies $eet_{\Omega(i,d_j) \cup S(i,d_j)} > d_j$.

Let $p, q \in \Omega$ be tasks with $r_p = r_\Omega$ and $d_q = d_\Omega$. Because of $lstmax_\Omega < d_i$ it holds $\Omega \subseteq \Omega(i)$. Moreover, let $S(i, r_p, d_q) = \{s \in S(i, d_q) \mid r_p \leq r_s\}$ be the set of pseudo-tasks that must be run in the time window $[r_p, d_q)$. Due to Def. 6 – amongst others the earliest start and latest end times of tasks in $\Omega(i)$

also are the earliest start and latest end times of the pseudo-tasks – it holds $e_{S(i,r_p,d_q)} = c_i(d_q - \max(\text{lst}_i, r_p))$. From this it follows that

$$\begin{aligned} d_q < r_p + \frac{e_\Omega + c_i(d_q - \max(\text{lst}_i, r_p))}{C} &= r_p + \frac{e_\Omega + e_{S(i,r_p,d_q)}}{C} \\ &\leq \text{eet}_{\Omega \cup S(i,r_p,d_q)} &\leq \text{eet}_{\Omega(i) \cup S(i,d_q)}. \end{aligned}$$

Hence, the inequality holds for the task q .

“ \Leftarrow ”: Let $j \in \Omega(i)$ be a task that satisfies the inequality $\text{eet}_{\Omega(i,d_j) \cup S(i,d_j)} > d_j$. We have to show the existence of a subset $\Omega \subseteq T \setminus \{i\}$ such that the condition of the not-last rule is satisfied.

Let $\Psi = \Omega(i, d_j) \cup S(i, d_j)$ be the considered task set and $\Psi' \subseteq \Psi$ be its subset that determines eet_Ψ , i.e. $\text{eet}_\Psi = r_{\Psi'} + e_{\Psi'}/C$. Thanks to Lemma 1 it follows $\Psi' = \{t \in \Psi \mid r_{\Psi'} \leq r_t\}$. Thus, $S(i, r_{\Psi'}, d_j)$ must be a subset of Ψ' and $\Psi' \setminus S(i, r_{\Psi'}, d_j)$ must be a subset of $\Omega(i, d_j)$. From this it follows that

$$\begin{aligned} d_j < \text{eet}_\Psi = r_{\Psi'} + \frac{e_{\Psi'}}{C} &= r_{\Psi'} + \frac{e_{\Psi' \setminus S(i,r_{\Psi'},d_j)} + e_{S(i,r_{\Psi'},d_j)}}{C} \\ &= r_{\Psi'} + \frac{e_{\Psi' \setminus S(i,r_{\Psi'},d_j)} + c_i(d_j - \max(\text{lst}_i, r_{\Psi'}))}{C} \end{aligned}$$

because there is a task $s \in S(i, r_{\Psi'}, d_j)$ (due to Def. 6) and a task $t \in \Psi'$ with $r_s = r_t = r_{\Psi'}$. Therefore, the not-last rule holds for i and $\Psi' \setminus S(i, r_{\Psi'}, d_j)$. \square

For efficiency reasons, the earliest end time of a task set Ψ is calculated by a binary balanced tree as proposed by Vilím, here called the Ψ -tree. There, a task $j \in \Psi$ is represented by a node containing the values $E_{V(j)}$ and $EET_{V(j)}$ which are derived from its children, and where $V(j)$ denotes the task set of all tasks in the subtree rooted at j and j itself. The nodes in Ψ are sorted in non-increasing order according to the earliest start times of their tasks.³ $r_s \leq r_j$ holds for each task s in the left subtree of a task j and $r_j \geq r_t$ holds for each task t in the right subtree. The values E and EET are derived as follows:

$$\begin{aligned} E_{V(j)} &:= E_{V(\text{left}(j))} + e_j + E_{V(\text{right}(j))} \\ EET_{V(j)} &:= \max\{ EET_{V(\text{left}(j))} + (e_j + E_{V(\text{right}(j))})/C, \\ &\quad r_j + (e_j + E_{V(\text{right}(j))})/C, \\ &\quad EET_{\text{right}(j)} \} , \end{aligned}$$

where $\text{left}(j)$ ($\text{right}(j)$) denotes the left (right) child of j . If j has no left (right) child, i.e. $\text{left}(j) = \text{nil}$ ($\text{right}(j) = \text{nil}$), then $E_{V(\text{nil})} = 0$ and $EET_{V(\text{nil})} = -\infty$ holds by convention. The Ψ -tree for the CuSP in Ex. 3 is shown in Fig. 4.

Theorem 2. *Let Ψ be a set of tasks and C be a resource capacity. Then for the values E_Ψ and EET_Ψ of the Ψ -tree it holds $E_\Psi = e_\Psi$ and $EET_\Psi = \text{eet}_\Psi$.*

Proof. Let Ψ be a set of n tasks and $w \in \Psi$ the root of the Ψ -tree. This means that it holds $E_{V(w)} = E_\Psi$ and $EET_{V(w)} = EET_\Psi$. The theorem is proved by induction over the number of tasks in Ψ .

³ In the following, we do not distinguish between tasks and their corresponding nodes.

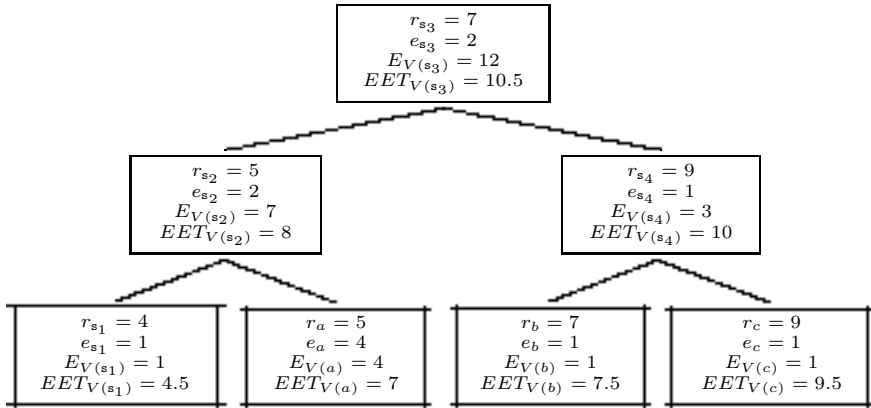


Fig. 4. A Ψ -tree for the tasks $\{a, b, c\}$ and $C = 2$ (cf. Ex. 3) including the pseudo-tasks regarding the task i . A visualization of $EET_{V(s_3)} = eet_{\Psi} = 10.5$ is shown in Fig. 3.

Base case: $n = 1$. The sole task in Ψ is the root w . Due to the definition of E and EET it follows $E_{\Psi} = e_w = e_{\Psi}$ and $EET_{\Psi} = r_w + e_w/C = eet_{\Psi}$.

Induction step: $n \rightarrow n + 1$. The precondition is the validity of the theorem for task sets with at most n elements.

Let Ψ be a task set with $|\Psi| = n + 1$, $\Psi' \subseteq \Psi$ with $eet_{\Psi} = r_{\Psi'} + e_{\Psi'}/C$, and $t \in \Psi'$ be the left-most task in the Ψ -tree from Ψ' , i.e. $\Psi' \cap V(\text{left}(t)) = \emptyset$ and $\nexists t' \in \Psi' : t \in V(\text{right}(t'))$, with $r_t = r_{\Psi'}$.

At first we prove $E_{\Psi} = e_{\Psi}$ and then $EET_{\Psi} = eet_{\Psi}$. According to the calculation formula and the precondition it follows that

$$\begin{aligned} E_{\Psi} &= E_{V(w)} = E_{V(\text{left}(w))} + e_w + E_{V(\text{right}(w))} \\ &= e_{V(\text{left}(w))} + e_w + e_{V(\text{right}(w))} = e_{\Psi}. \end{aligned}$$

Given that the Ψ -tree is sorted by the earliest start times it holds that $r_{V(\text{left}(w))} \leq r_w \leq r_{V(\text{right}(w))}$. The equality of $EET_{\Psi} = eet_{\Psi}$ is proved in two cases.

$EET_{\Psi} \leq eet_{\Psi}$: Due to $r_{V(\text{left}(w))} \leq r_w \leq r_{V(\text{right}(w))}$ and the precondition the following inequalities are satisfied.

$$\begin{aligned} EET_{V(\text{left}(w))} + \frac{e_w + E_{V(\text{right}(w))}}{C} &= eet_{V(\text{left}(w))} + \frac{e_w + e_{V(\text{right}(w))}}{C} \\ &= eet_{V(\text{left}(w)) \cup \{w\} \cup V(\text{right}(w))} \leq eet_{\Psi} \text{ ,} \\ r_w + \frac{e_w + E_{V(\text{right}(w))}}{C} &= r_w + \frac{e_w + e_{V(\text{right}(w))}}{C} \\ &= eet_{\{w\} \cup V(\text{right}(w))} \leq eet_{\Psi} \text{ ,} \\ EET_{V(\text{right}(w))} &= eet_{V(\text{right}(w))} \leq eet_{\Psi} \text{ .} \end{aligned}$$

From these inequalities it follows that $EET_{\Psi} \leq EET_{V(w)} \leq eet_{\Psi}$.

$EET_{\Psi} \geq eet_{\Psi}$: If $t \in V(\text{left}(w))$ is in the left subtree of the root w then $\{w\} \cup V(\text{right}(w)) \subseteq \Psi'$ (cf. Lemma [1](#)), $r_{V(\text{left}(w))} \leq r_t \leq r_w \leq r_{V(\text{right}(w))}$ and the precondition holds for $V(\text{left}(w))$. Let $L' := \Psi' \setminus (\{w\} \cup V(\text{right}(w)))$ be the task set of Ψ' in the left subtree rooted at w . Now, the following applies:

$$\begin{aligned} eet_{\Psi} &= r_{\Psi'} + e_{\Psi'}/C = r_{L'} + (e_{L'} + e_{\{w\} \cup V(\text{right}(w))})/C \\ &= eet_{L'} + e_{\{w\} \cup V(\text{right}(w))}/C \\ &\leq eet_{V(\text{left}(w))} + e_{\{w\} \cup V(\text{right}(w))}/C \\ &= EET_{V(\text{left}(w))} + (e_w + E_{V(\text{right}(w))})/C \leq EET_{\Psi} . \end{aligned}$$

If $t = w$ is the root of the Ψ -tree then $\Psi' = \{w\} \cup V(\text{right}(w))$ thanks to Lemma [1](#) and it follows:

$$\begin{aligned} eet_{\Psi} &= r_{\Psi'} + e_{\Psi'}/C = r_w + (e_w + e_{V(\text{right}(w))})/C \\ &= r_w + (e_w + E_{V(\text{right}(w))})/C \leq EET_{\Psi} . \end{aligned}$$

If $t \in V(\text{right}(w))$ holds, $\Psi' \subseteq V(\text{right}(w))$ holds due to the choice of t . From the precondition it follows

$$eet_{\Psi} = r_{\Psi'} + e_{\Psi'}/C = EET_{V(\text{right}(w))} \leq EET_{\Psi} .$$

Therefore, the hypothesis is shown. □

Based on the results shown in Theorems [1](#) and [2](#) we are able to formulate the L-N2L algorithm. It is presented in Alg. [1](#) and works as follows: The outer loop (lines 2–16) iterates over all tasks $i \in T$ and initializes the Ψ -tree by the empty tree, $lstmax_{\Psi}$ by $-\infty$, z by 1, and $ptime$ by lst_i in the first steps (line 3). The last two variables are used to build the pseudo-tasks and $lstmax_{\Psi}$ stores the latest start time of a task j in the Ψ -tree. The inner loop (lines 4–16) iterates over the array Y sorted in non-decreasing order of the latest end times. It extends stepwise the Ψ -tree by tasks from $\Omega(i)$ (lines 5–6) and the corresponding pseudo-tasks s (line 10) which are built in a while loop (8–13). Hence, the nodes in the Ψ -tree are equal to the set $\Omega(i, d_{Y[y]}) \cup S(i, d_{Y[y]})$ in the y -th iteration after the insertion of the tasks and pseudo-tasks (after line 13).

In line 14 the existence of a subset of $\Omega(i, d_{Y[y]})$ that satisfies with i the not-last rule is checked by the inequality $EET_{\Psi} > d_{Y[y]}$. If the inequality holds then the “new” latest end time $UB'[i]$ is replaced by $\min(lstmax_{\Psi}, UB'[i])$, and the inner loop terminates; Otherwise, the algorithm remains in the inner loop.

Theorem 3. *The L-N2L is sound, its time complexity is $\mathcal{O}(n^2 \log n)$ and its space complexity is $\mathcal{O}(n)$.*

Proof (Theorem [3](#)). At first we show the soundness and then the complexity.

Soundness: Here, it means that the algorithm calculates any new upper bound $UB'[i]$ for each task $i \in T$ given the CuSP instance (T, C) as follows:

$$UB[i] \leq UB'[i] < d_i, \text{ if } UB[i] < d_i \quad \text{and} \quad UB'[i] = d_i, \text{ otherwise ,}$$

where $UB[i]$ is the lowest upper bound for i concerning the not-last rule.

Algorithm 1. L-N2L Applies the not-last rule in $\mathcal{O}(n^2 \log n)$.

```

Input :  $Y$  an array of tasks sorted in non-decreasing order of the latest end
          times ( $d_{Y[1]} \leq \dots \leq d_{Y[n]}$ ).
Input :  $Z$  an array of the tasks' earliest start and latest end times sorted in
          non-decreasing order of the time points ( $Z[i] < Z[i + 1]$ ).
Private:  $\Psi$  a balanced binary tree sorted by the tasks' earliest start times.
Result:  $UB'$  an array of the new upper bounds on the task end time.
1 foreach  $i \in T$  do  $UB'[i] := d_i$ ;
2 for  $i \in T$  do
3    $\Psi := \emptyset$ ,  $lstmax_\Psi := -\infty$ ,  $z := 1$ ,  $pretime := lst_i$ ;
4   for  $y := 1$  to  $n$  do
5     if  $Y[y] \in \Omega(i)$  then
6       insert  $Y[y]$  in  $\Psi$ -tree with value  $r_{Y[y]}$ ;
7        $lstmax_\Psi := \max(lst_{Y[y]}, lstmax_\Psi)$ ;
8       while  $z \leq Z.length$  and  $Z[z] \leq d_{Y[y]}$  do
9         if  $Z[z] > lst_i$  then
10          build the pseudo-task  $s$  concerning  $i$  by  $r_s := pretime$ ,
11           $d_s := Z[z]$ ,  $c_s := c_i$ , and  $p_s := d_s - r_s$ ;
12          insert  $s$  in  $\Psi$ -tree with value  $r_s$ ;
13           $pretime := Z[z]$ ;
14           $z := z + 1$ ;
15         if  $EET_\Psi > d_{Y[y]}$  then
16            $UB'[i] := \min(lstmax_\Psi, UB'[i])$ ;
17           break;
18 foreach  $i \in T$  do  $d_i := UB'[i]$ ;

```

The algorithm initializes the array UB' in line 1 by the latest end times of all tasks. Because the value $UB'[i]$ is only updated by $\min(lstmax_\Psi, UB'[i])$ (line 15) it follows that $UB'[i] \leq d_i$.

Apparently, if the algorithm is in its i -th iteration of the outer loop and y -th iteration of the inner loop with $Y[y] \in \Omega(i)$ before line 14 then it holds that the Ψ -tree contains the tasks in $\Omega(i, d_{Y[y]}) \cup S(i, d_{Y[y]})$. Therefore, it checks all necessary inequalities $eet_{\Omega(i, d_j) \cup S(i, d_j)} > d_j$ ($j \in \Omega$) for the not-last rule in the line 14 due to the Theorems 1 and 2. If all inequalities are not satisfied then the algorithm will not compute a new latest end time for i and thus $UB'[i] = d_i$ holds. Otherwise, let y' be the first iteration in the inner loop that satisfies the inequality. In this case the algorithm determines the new latest end time by $\min(lstmax_\Psi, UB'[i])$ where $lstmax_\Psi = lstmax_{\Omega(i, d_{Y[y']})}$. Because of the initialization of $lstmax_\Psi$ by $-\infty$ and its update by $\max(lst_{Y[y]}, lstmax_\Psi)$ for $Y[y] \in \Omega(i, d_{Y[y']})$ (line 7) it holds that $UB'[i] \leq lstmax_{\Omega(i, d_{Y[y']})} < d_i$. Hence, the algorithm is sound.

Complexity: The time complexity $\mathcal{O}(n^2 \log n)$ is determined by the outer loop (lines 2–16) whereas the complexity for the sorting of the arrays Y and Z , the initialization of UB' (line 1), and the update of the latest end times (line 17) are negligible.

The outer loop iterates over all tasks $i \in T$. For each task i a Ψ -tree is generated. Before the loop the Ψ -tree is empty. During the loop it is extended stepwise by tasks and pseudo-tasks which all are inserted once in the Ψ -tree.

The pseudo-tasks are built in a while loop (lines 8–13) that iterates over the variable z which is initialized with 1 before the inner loop. At each iteration of the while loop z is incremented by 1, so that the while loop is traversed at most $2n$ for i . Hence, at most $2n$ pseudo-tasks are inserted in the Ψ -tree.

Therefore, the Ψ -tree contains at most $3n$ tasks. As the result of that an insertion and a deletion cost $\mathcal{O}(\log n)$ in the tree. All other operations in the algorithm can be performed in constant time. Therefore, the time complexity of the inner loop is $\mathcal{O}(n \log n)$ and the overall time complexity of the algorithm $\mathcal{O}(n^2 \log n)$. Obviously, the space complexity of the algorithm is $\mathcal{O}(n)$. \square

Theorem 4. L-N2L computes the lowest upper bounds $UB[i]$ concerning the not-last rule for each $i \in T$ after at most $|\{lstmax_j \mid j \in T\}|$ iterations.

Proof. In the following we first prove by contradiction that L-N2L computes the upper bounds $UB[i]$ concerning the not-last rule for each $i \in T$ after a finite number of iterations. Therefore, we assume that after k iterations of L-N2L not any due date d_i is updated in line 17 but there is a task $c \in T$ such that $UB[c] < d_c$ holds. Thus any further iteration will result in the same situation, i.e. at least one lowest upper bound will be not reached, because the input of the deterministic L-N2L will not change after the k -th iteration. However, there is a task set $\Omega \subseteq \Omega(c)$ satisfying the not-last rule such that $lstmax_\Omega = UB[c]$ holds. According to Theorem 1 there is another task $j \in \Omega(c)$ satisfying $eet_{\Omega(c,d_j) \cup S(c,d_j)} > d_j$ where $S(c, d_j) := \{s \in S(c) \mid d_s \leq d_j\}$. This means that in the $k+1$ iteration of L-N2L the bound $UB'[c]$ is updated (cf. line 15) such that $UB'[c] \leq lstmax_{\Omega(c,d_j)} < d_c$ holds by definition of $\Omega(c)$ changing d_c eventually. Obviously this contradicts the assumption that after k iterations a fixed point is reached. Further, these considerations also show that each d_i will be improved in each iteration of L-N2L as long as the lowest upper bound is not reached. Thus, $UB[i] = d_i$ holds for each $i \in T$ after at most $|\{lstmax_j \mid j \in T\}|$ iterations. \square

It follows that the time complexity for the computation of the best upper bounds using L-N2L is $\mathcal{O}(n^3 \log n)$ in the worst case and thus equal to the time complexity of the currently most efficient, complete not-last algorithm [14].

4 Experiments

Experiments were carried out in order to compare complete and incomplete algorithms on the numbers of iterations necessary for fixed point computations and on their absolute runtimes for problems of different sizes. The latter indicates if and for which problem sizes the reduced complexity pays off. Therefore, examinations were performed on RCPSP⁴ from the well-established benchmark library PSPLib [7] and the library of Baptiste and LePape [1] (BL). The firstCS constraint solver [6] was used to evaluate the different filtering algorithms.

⁴ RCPSP consists of a set of tasks requiring one or more resources for their execution, precedences between tasks and a set of renewable resources.

Table 1. Comparison between different not-first/not-last algorithms

solved	BL20		BL25		J30		J60		J90	
	time	#it	time	#it	time	#it	time	#it	time	#it
L-N2L	no	18.1 34016	30.9 37987	69.2 36776	157.7 29040	276.2 23419				
	yes	3.5 6344	2.2 3084	0.8 652	6.5 1078	22.3 1611				
L-N3	no	9.9 34002	19.5 37956	57.8 36757	158.6 29028	333.7 23391				
	yes	1.9 6343	1.3 3082	0.6 652	6.1 1078	25.1 1611				
C-N3L	no	19.2 33941	38.3 37901	100.3 36666	344.3 28922	*842.1 *23203				
	yes	3.8 6326	2.8 3082	1.1 652	14.8 1078	69.6 1611				

In order to compare the number of iterations for the different not-first/not-last algorithm an upper limit on the number of backtracks (bts) of 5000 (20000) was imposed for PSPLib (BL) instances. This limit also has the advantage that the main runtime differences can be only caused by the algorithms, once it was hit. The number of the limits were chosen in this way that the algorithms were sufficiently often iterated and hit the bts limit within one hour runtime for almost all instances. Moreover, the search was stopped after one hour if it did not reach the bts limit. The time limit was only reached in some cases.

A cumulative resource in RCPSP was modeled by one cumulative propagator for all tasks requiring this resource and one disjunctive propagator for all tasks requiring more than the half of the resource capacity. Both propagators are idempotent, i.e. the execution of their filtering algorithms is repeated until no further pruning is performed. The order of the algorithms for the cumulative propagator was chosen as follows: time-table, edge-finding, one of the examined implementations of the considered not-first/not-last algorithms and overload checking. This sequence was repeated until the fixed point was reached.

A (standard) dichotomic branch&bound optimization was applied to find the minimal makespan, i.e. the smallest end time for the project. The underlying search selected the task with the smallest start time in its start domain. If there is a tie between several tasks then the task with the smallest latest start time is picked. If a tie remains, the first task in the order of occurrence is selected. Before the smallest start time is assigned to the start variable of the selected task a choicepoint is created. If the propagation or the further search fails then the search backtracks to the recently created choicepoint and the smallest start time that yields the failure is excluded from the variable's domain.

Table 1 compares the average time in seconds either for reaching the bts limit or for finding a solution, and the average number of the iteration (#it) of the not-first/not-last algorithm. For table entries that are marked by "*" the time limit was hit before the bts limit for some instances. The classes J30, J60, and J90 are from the PSPLib containing 480 instances with 30, 60, and 90 tasks respectively, and the classes BL20, and BL25 are from Baptiste and Le Pape containing 20 instances with 20, and 25 tasks respectively. We compared the new, incomplete algorithm L-N2L with the corrected, but still incomplete $\mathcal{O}(n^3)$ algorithm L-N3 and with the complete $\mathcal{O}(n^3 \log n)$ algorithm C-N3L both presented in [14].

The algorithm L-N3 is the fastest for the considered instances with up to 60 tasks. For the instances with 90 tasks the algorithm L-N2L⁵ is faster. The complete algorithm C-N3L is always inferior. One reason is that the new incomplete algorithm only iterate less than 1% more than the complete algorithm. The other reason is that the time complexity of C-N3L is higher than L-N2L and LN3 due to its completeness.

We also run the experiments without disjunctive constraints, edge-finding, and/or not-first/not-last. The results with not-first/not-last were very similar to the results in Tab. 1. The results without not-first/not-last were inferior in terms of the number of found solutions within the given limits. The reader is encouraged to see the detailed results in [11].

5 Conclusion and Future Work

In this paper we presented a new pruning algorithm concerning the not-first/not-last rule for cumulative resources in time complexity $\mathcal{O}(n^2 \log n)$ and space complexity $\mathcal{O}(n)$. Its soundness and its complexity were proved. This new algorithm is incomplete, i.e. it might not find the best pruning at once, but at least one. We compared this algorithm with (all) other not-first/not-last algorithm on RCPSP instances from the PSPLib and Baptiste and Le Pape. The results show that the new algorithm is superior to a complete not-first/not-last algorithm already for small instances and to an incomplete not-first/not-last algorithm starting from problems with 60 tasks.

Future work will concentrate on finding a complete algorithm with $\mathcal{O}(n^2 \log n)$ time complexity which can be important e.g. for building minimal explanations of time bound adjustments in a constraint system using no-good learning like the lazy clause generator [10]. Recently, the time complexity for pruning based on edge-finding dropped to $\mathcal{O}(kn \log n)$ [17] which indicates that there might be a more efficient not-first/not-last algorithm possibly with the same complexity.

Acknowledgments. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

References

1. Baptiste, P., Le Pape, C.: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints* 5(1-2), 119–139 (2000)
2. Baptiste, P., Le Pape, C., Nuijten, W.: *Constraint-Based Scheduling*. International Series in Operations Research & Management Science, vol. 39. Kluwer Academic Publishers, Dordrecht (2001)
3. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. *Mathematical and Computer Modelling* 12, 97–123 (1994)

⁵ The performance of the not-first/not-last algorithms can be improved if a pre-sorted array is used to represent the Ψ -tree instead of the AVL tree used in the experiments.

4. Blazewicz, J., Lenstraand, J.K., Rinnooy Kan, A.H.G.: Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* 5, 11–24 (1983)
5. Caseau, Y., Laburthe, F.: Cumulative scheduling with task intervals. In: Maher, M.J. (ed.) *Joint International Conference and Symposium on Logic Programming – JICSLP 1996*, pp. 363–377. MIT Press, Cambridge (1996)
6. Hoche, M., Müller, H., Schlenker, H., Wolf, A.: **firstCS** - A Pure Java Constraint Programming Engine. In: Hanus, M., Hofstedt, P., Wolf, A. (eds.) *2nd International Workshop on Multiparadigm Constraint Programming Languages – MultiCPL 2003* (September 2003), uebb.cs.tu-berlin.de/MultiCPL03/Proceedings.MultiCPL03.RCoRP03.pdf
7. Kolisch, R., Sprecher, A.: PSPLIB – A project scheduling problem library. *European Journal of Operational Research* 96(1), 205–216 (1997)
8. Mercier, L., Van Hentenryck, P.: Edge finding for cumulative scheduling. *INFORMS Journal on Computing* 20(1), 143–153 (2008)
9. Nuijten, W.P.M.: *Time and Resource Constrained Scheduling*. PhD thesis, Eindhoven University of Technology (1994)
10. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. *Constraints* 14(3), 357–391 (2009)
11. Schutt, A.: *Entwicklung suchraumeinschränkender Verfahren zur constraint-basierten Lösung kumulativer Ressourcenplanungsprobleme*. In: Diplomarbeit, Humboldt Universität zu Berlin, Berlin, Germany (September 2006) (in German)
12. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Explaining the cumulative propagator. To appear in *Constraints*
13. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Why cumulative decomposition is not as bad as it sounds. In: Gent, I.P. (ed.) *CP 2009*. LNCS, vol. 5732, pp. 746–761. Springer, Heidelberg (2009)
14. Schutt, A., Wolf, A., Schrader, G.: Not-first and not-last detection for cumulative scheduling in $\mathcal{O}(n^3 \log n)$. In: Umeda, M., Wolf, A., Bartenstein, O., Geske, U., Seipel, D., Takata, O. (eds.) *INAP 2005*. Schutt, A., Wolf, A., Schrader, G., vol. 4369, pp. 66–80. Springer, Heidelberg (2006)
15. Simonis, H., O’Sullivan, B.: Search strategies for rectangle packing. In: Stuckey, P.J. (ed.) *CP 2008*. LNCS, vol. 5202, pp. 52–66. Springer, Heidelberg (2008)
16. Vilím, P.: $\mathcal{O}(n \log n)$ filtering algorithms for unary resource constraint. In: Régim, J.-C., Rueher, M. (eds.) *CPAIOR 2004*. LNCS, vol. 3011, pp. 335–347. Springer, Heidelberg (2004)
17. Vilím, P.: Edge finding filtering algorithm for discrete cumulative resources in $\mathcal{O}(kn \log n)$. In: Gent, I.P. (ed.) *CP 2009*. LNCS, vol. 5732, pp. 802–816. Springer, Heidelberg (2009)
18. Vilím, P., Barták, R., Čepeck, O.: Unary resource constraint with optional activities. In: Wallace, M. (ed.) *CP 2004*. LNCS, vol. 3258, pp. 62–76. Springer, Heidelberg (2004)
19. Wolf, A., Schrader, G.: $\mathcal{O}(kn \log n)$ overload checking for the cumulative constraint and its application. In: Umeda, M., Wolf, A., Bartenstein, O., Geske, U., Seipel, D., Takata, O. (eds.) *INAP 2005*. LNCS (LNAI), vol. 4369, pp. 88–101. Springer, Heidelberg (2006)

A Generic Visualization Platform for CP

Helmut Simonis¹, Paul Davern¹, Jacob Feldman¹,
Deepak Mehta¹, Luis Quesada¹, and Mats Carlsson^{2,*}

¹ Cork Constraint Computation Centre

Department of Computer Science, University College Cork, Ireland

² Swedish Institute of Computer Science

SICS AB, Uppsala Science Park, SE-751 83 Uppsala, Sweden

`h.simonis@4c.ucc.ie`

Abstract. In this paper we describe the design and implementation of CP-VIZ, a generic visualization platform for constraint programming. It provides multiple views to show the search tree, and the state of constraints and variables for a post-mortem analysis of a constraint program. Different to most previous visualization tools, it is system independent, using a light-weight, intermediate XML format to exchange information between solvers and the visualization tools. CP-VIZ is available under an open-source licence, and has already been interfaced to four different constraint systems.

1 Introduction

Visualization¹ is one of the best techniques for understanding the behavior of constraint programs, allowing us to directly observe the impact of changes by visual inspection instead of using tedious debugging. So far, most constraint visualization tools have been closely linked to specific solvers, making it difficult to compare alternative solvers and to reuse development effort spent on other systems. Previous attempts [4] at generic tools did not find widespread use largely due to the complexity of the specification and the level of detail captured. The new, light-weight CP-VIZ system provides a simple XML based interface for solvers, and can be easily extended for new systems and constraints. In CP-VIZ, we try to visualize the search tree and the state of variables and (global) constraints in parallel views. The search tree shows choices, assignments and failures, modeled on the tree display in the Oz Explorer [14] and later in CHIP [16]. Constraints and variables are shown in a 2D layout defined by the user, individual global constraints are shown in custom visualizations similar to [17]. A new constraint can be added to the package by simply deriving a new class with a custom drawing method.

* This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886). The support of Cisco Systems and of the Silicon Valley Community Foundation is gratefully acknowledged.

¹ Visualization relies heavily on the use of colors, with a potential loss of information if seen in black&white only. An on-line version of the paper with colored diagrams can be downloaded from the URL <http://4c.ucc.ie/~hsimonis/cpviz.pdf>. Also note that in the electronic version you can zoom into all SVG diagrams, revealing additional information.

The design of the visualization tool was driven by user-requirements, coming mainly from the development of an ECLiPSe ELearning course.

Visualization has played a significant role in demonstrating the use of constraint programming, and helping to develop successful applications. Systems like CHIP [6] relied on Prolog-based coroutines to visualize the assignment of variables changing throughout a search process. Visualizations were written as application specific tools which were co-developed with the constraint model. This approach restricted re-use of components and was tightly linked to a logic-programming host language. Meier [12] was the first to abstract visualization types based on collections of variables and to propose different views for them. The visualization of the search tree was pioneered in the Oz Explorer [14], its interactive use tightly linked to the generalized branching possibilities of the Oz environment. The DISCiPl project [5] produced a multitude of results for constraint debugging and visualization, the ones most relevant for this paper are the search tree tool for CHIP [16] and the idea of specialized visualizers for global constraints [17]. The French OADymPPaC project [4] considered a system independent view of visualization. But the XML-based specification for post-mortem traces was quite complex and achieved limited acceptance, and seems no longer to be actively maintained. The main design aim for the OADymPPaC trace format was to capture all possible information about program execution. The visualization tools would then extract those pieces which were of interest to them. While this allowed different tools to work at different abstraction levels, it also required rather deep integration into each supported CP solver to generate the trace, and led to very large trace files for even relatively small problems. The visualization tools for Comet [7] provide an environment for developing visualizations for constraint-based local search, which mix generic and application specific aspects of the visualization inside the modeling language.

2 Design Aims

The design of CP-VIZ was largely driven by the development of an ECLiPSe ELearning course [15], for which we wanted to be able to show and explain the solving of various models for application programmers. We did not want to restrict the use of the visualizer to ECLiPSe only, but rather tried to design a constraint system independent architecture. This led to a number of key design decisions:

- We decided to concentrate on post-mortem analysis, which minimizes the requirements of interaction between the constraint solver and the visualization environment, but still provides most of the information required for analysis.
- The output of the visualization can be studied on-screen, but can also be provided as high-quality, colored, vector based print output. Data feeds for other visualization tools are also provided.
- The tools are solver independent, written in a general purpose language (Java) and can be easily extended and specialized by deriving new classes from existing visualization classes.
- We added invariant checking at each search node to the functionality, this allows a solver independent validation of the results, and can highlight missing propagation in individual constraints.

- The system is platform independent, and is provided as open source. The system specific XML generation requires minimal effort.

At the same time, these choices restricted some functionality that is provided in other visualization tools [14,16].

- The tool currently is not designed for interactive problem solving, controlling the search from within the visualization by making manual decisions on how the search should progress. To allow this in a system independent way seems quite difficult, and would require rather deep interaction with each solver. At the same time, there is limited evidence that such an interactive use helps application programmers in developing strategies which solve problems automatically.
- We are not considering individual propagation steps, showing in which order constraints are woken and how they make detailed domain restrictions. For most application programmers, this level of abstraction is too detailed, and it is too time consuming to follow the execution of constraint propagators through a longer search process.
- We don't collect and display the constraint graph. Many existing tools for displaying constraint graphs [9,8] seem to work only for binary constraints, and heavily rely on graph layout algorithms, without finding usable displays for larger problem sizes.

Which type of design choices can be improved when using visualization? Figure 1 shows the well-known example of a Sudoku puzzle expressed in constraint programming, which compares three different consistency levels for the ALLDIFFERENT constraints in the model. The same problem is modeled using forward checking, bounds consistency and domain consistency; the pictures show the state after the initial set-up, before search is started. The variables are shown in form of a two-dimensional matrix, each cell corresponds to a variable, which shows the current values in the domain (small, in green) or the assigned value (large, in red). For this carefully selected, didactic example, different consistency levels lead to different amounts of propagation, but this is not universally true. In many applications the visualization can help to decide

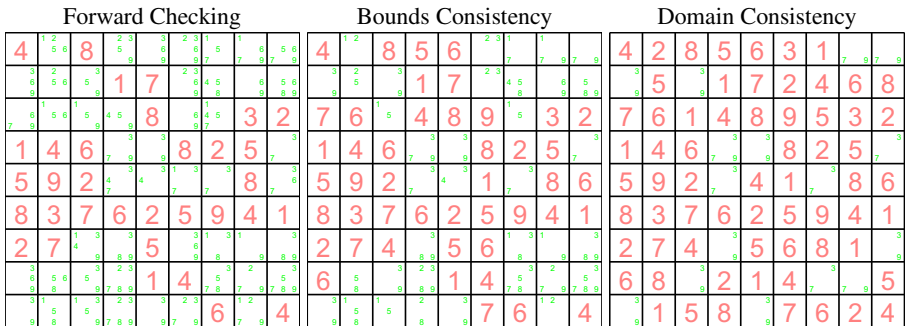


Fig. 1. Sudoku: Consistency Level Comparison

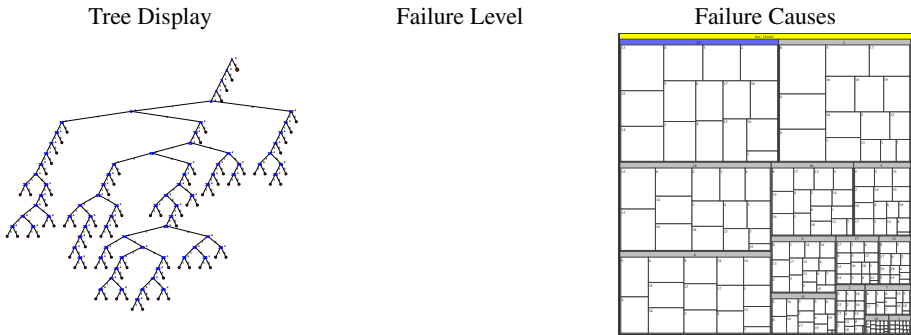


Fig. 2. Search Tree Analysis - Different Views of Search Tree Data

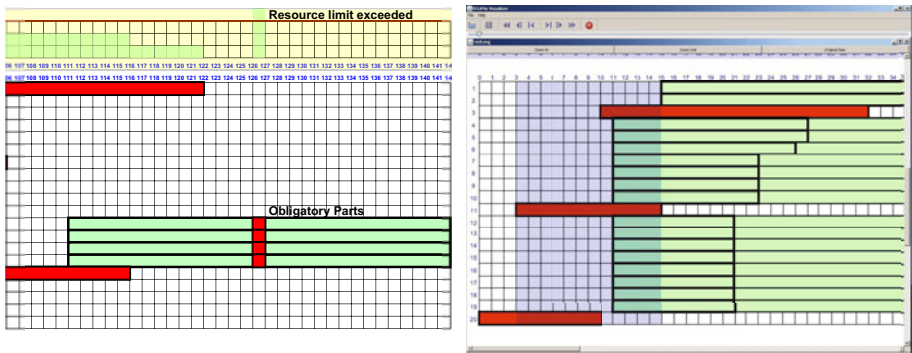


Fig. 3. Invariant Checks for Cumulative Scheduling Problem

which consistency level to use in order to find the right compromise between speed, propagation and problem solving stability.

Figure 2 shows different diagrams for visualization of the search tree. If the search space is small, the full tree can be shown (on the left). For more complex problems, this is no longer possible, and a more compact form, originally proposed in [14], which abstracts failed sub-trees, can be displayed (see Figure 7 for an example). But often this detailed analysis is not required, it suffices to have a simple quantitative analysis, as shown in the middle part of Figure 2. It plots the number of success and failure nodes with the depth of the search tree. The shape of the plot often is enough to understand how well a model is able to explore the search space. On the right we show a treemap visualization which indicates the size of the generated subtree below a top-level choice in the search. This can help to understand more clearly if the search strategy globally is making the right choices.

Finally, the diagrams in Figure 3 show an example where invariant checking was used to detect nodes in the search where the constraint propagation was not sufficient. The pictures are from a cumulative scheduling problem proposed by Robert

Nieuwenhuis [13] solved in ECLiPSe. It highlights two problems with the CUMULATIVE constraint implementation of ECLiPSe, which is based on edge finding. In the left picture, some tasks in a partial assignment are restricted sufficiently so that obligatory parts (dark, in red) are generated. The sum of these obligatory parts exceeds the resource limit, which is not detected by the propagator. Invariant checking highlights the constraint and has also marked the problem in the search tree. On the right, a number of tasks have been assigned, and their resource profile reaches the resource limit, but the start times of unassigned tasks are not updated properly. This not only shows some missing propagation, but affects the search routine as well, as the heuristic for task selection will pick the wrong tasks to be assigned next. The problem was resolved by developing another propagator for CUMULATIVE based on obligatory parts.

3 Architecture

Figure 4 shows the basic architecture of the CP-VIZ system. The visualization is driven by annotations in the constraint program. When run in the solver, two XML log files (one for the search tree, the other for the constraint and variable visualization) are produced. These files are then parsed in the main CP-VIZ application, producing graphical output as SVG, or as input for other tools (tree maps, graphs, statistics). The SVG output can be displayed interactively in the CP-VIZTOOL, or can be used in multiple ways to produce annotated or converted output for print or WEB media.

We use XML text files to link the generation of the log files to the creation of the visualization. This should allow almost any constraint programming system to be linked to the CP-VIZ visualization with minimal effort.

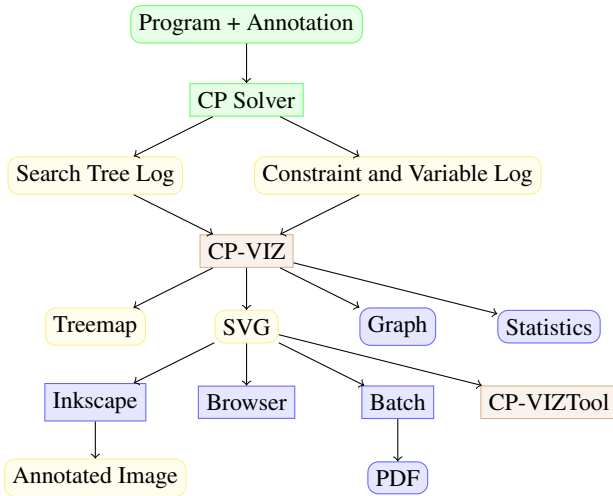


Fig. 4. CP-VIZ System Architecture

Search Tree Log. The log file consists of a single *tree* element, which contains a sequence of node elements which describe the search tree. There is an initial *root* node, which defines the start of the search, and *try* and *fail* nodes for successful and failed choices. In each node we have a node id, the node id of the parent node, the name of the variable currently assigned, the size of its domain, and the value assigned. A variant of these types also allows to handle arbitrary choices, not based on variable assignment. These alternatives can be useful to describe more complex branching schemes, but their analysis is slightly more restricted. A *solution* node is used to mark choice nodes which complete an assignment, i.e. to mark nodes where all constraints are satisfied. The format does not assume chronological depth first search, nodes can be added for any parent at any time.

Constraint and Variable Log. The second log file is used to describe snapshots of constraints and variables. Its top element is *visualization*, which contains a list of *visualizer* elements, describing the constraints and variables to be displayed. This is followed by a sequence of *state* elements, each containing a snapshot of the execution at a given time point. Inside each state, the *visualizer.state* elements describe the current state of a constraint or collection of variables. The syntax used roughly follows the syntax used in the global constraint catalog [3]. Constraints can be described by their named *arguments*, which may contain *collections* of basic types or *tuples*, which describe structures of disparate types. The basic types currently allowed are integers and finite domain variables, integer sets and domain variables over finite sets, plus some more specialized types.

3.1 System Dependent XML Generators

For every constraint system that wishes to use the CP-VIZ environment, we need to define an interface to generate the XML logs. Figure 5 shows such an interface for Java, based on two classes, *VisualSolver* and *VisualProblem*. The methods for the search tree log are contained in the *VisualSolver* interface, each adds or annotates a search node in the tree.

The methods for the *VisualProblem* class are split into two groups. The application programmer can use the method *register()* to register a constraint or a collection of variables with the visualization. There is also a method *snapshot()* which triggers the creation of a snapshot of all registered constraints and variables at a given program point. The snapshot is created by sending a *snapshot()* message to each registered constraint. This is then responsible for saving the current state of the constraint into the log. For this it might use the remaining methods of the *VisualProblem* class, which log XML elements of different types for the constraint.

3.2 CP-VIZ

The main CP-VIZ application parses the XML log files and creates SVG output for the user. The search tree is parsed completely before generation, while the constraint and variable snapshots are handled one at a time. In order to see which changes have occurred to the variables by the current search step, the tool keeps a stack of snapshots for all parents of the current node in memory. This not only allows to see the domain

```

public interface VisualSolver extends Visual {
    public void addRootNode(int id);
    public void addSuccessNode(int id, int parentId,
        String variableName, int size, int value);
    public void addSuccessNode(int id, int parentId,
        String variableName, int size, String choice);
    public void addFailureNode(int id, int parentId,
        String variableName, int size, int value);
    public void addFailureNode(int id, int parentId,
        String variableName, int size, String choice);
    public void labelSolutionNode(int id);
}

public interface VisualProblem extends Visual {
    public void register(Constraint constraint);
    public void register(Var var);
    public void register(Var[] varArray);
    public void register(Var[][] varMatrix);
    public void snapshot();

    // for implementors only
    public void startTagArgument(String index);
    public void startTagArgument(int index);
    public void endTagArgument();

    public void startTagCollection(String index);
    public void startTagCollection(int index);
    public void endTagCollection();

    public void startTagTuple(String index);
    public void startTagTuple(int index);
    public void endTagTuple();

    void tagVariable(Var var);
    void tagVariable(String index, Var var);
    void tagVariable(int index, Var var);

    void tagInteger(String index, int value);
    void tagInteger(int index, int value);
}

```

Fig. 5. VisualSolver and VisualProblem Interface Definition

updates of the variables, but also permits to generate path based visualizations [16], which display the evolution of a variable or some parameter through all parent nodes from the root to the current node.

Internally, the CP-VIZ application uses an event-based SAX-2 XML parser, so that it can minimize which part of the XML tree it needs to keep in memory. Experiments have shown that log files of several hundred Mb do not pose any problems.

3.3 CP-VIZ Tool

Figure 6 shows the CP-VIZTOOL, a Java application which displays the result of the visualization on the screen. The application has a time-line at the top, where the user can select a state of the execution for display. The tool will then display the state of the search tree in the left main pane, and the corresponding snapshot of the constraint and variable visualization in the right pane. The user can also step forward/backwards through the execution, or display the complete solution process as a movie, progressing automatically through the different snapshots.

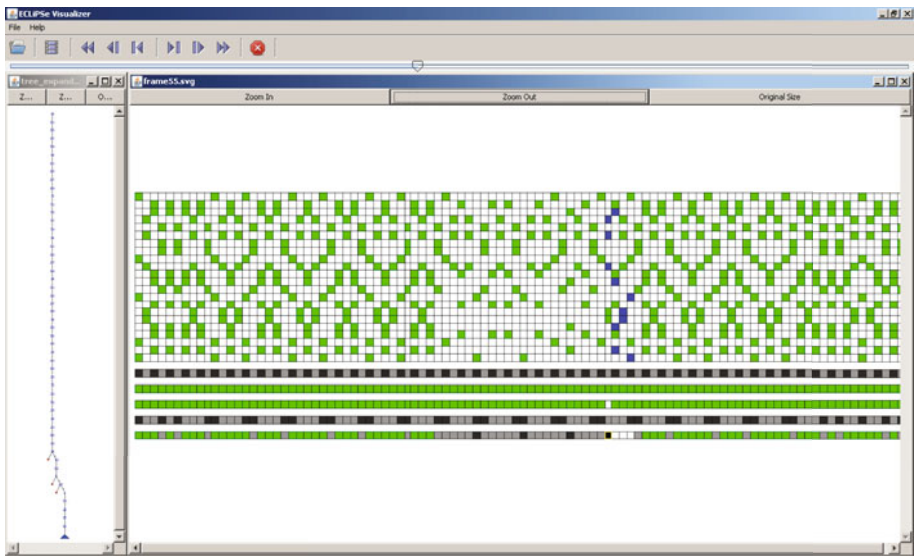


Fig. 6. Interactive CP-VIZ Tool for Car Sequencing Problem

4 Invariant Checking

By providing snapshots of the execution at fix points only, when all constraints have performed their consistency checking, CP-VIZ also provides data for systematic testing of execution traces. We have implemented an invariant checker, which for every snapshot calls an *invariant()* method for each registered constraint. This method may return *TRUE*, also the default value, or one of the values *INTERESTING*, *MISSING_PROPAGATION*, *INCONSISTENT* or *FALSE*. Combining all invariant checks for a snapshot, the visualizer then marks the node in the search tree accordingly and highlights any failed assertions in the constraint visualization. We explain the meaning of

the values for the example of a CUMULATIVE [11] constraint. The CUMULATIVE constraint states that the resource consumption of a collection of n tasks with start times $s_i \geq 0$, fixed duration d_i and resource use r_i must stay below the resource limit l and within the scheduling period p . A ground solution must satisfy the equations

$$\forall 0 \leq t < p : \sum_{\{i \mid s_i \leq t < s_i + d_i\}} r_i \leq l \quad (1)$$

$$\forall 1 \leq i \leq n : s_i + d_i \leq p \quad (2)$$

$$\sum_{1 \leq i \leq n} d_i * r_i \leq l * p \quad (3)$$

Inequality (3) is implied by the others, but is used as it provides a good basis for developing invariants. If for a ground instance one of these equations is not satisfied, then the invariant checker will return *FALSE*.

We can rewrite constraint (3) to consider upper bounds on domain variables l and p . This produces

$$\bar{p} \geq \left\lceil \frac{\sum d_i * r_i}{\bar{l}} \right\rceil \quad (4)$$

If in any snapshot this invariant does not hold, then the snapshot is inconsistent, i.e. the constraint propagator should have failed for this node. The invariant checker returns *INCONSISTENT*. A weaker invariant checks the lower bound of p instead:

$$\underline{p} \geq \left\lceil \frac{\sum d_i * r_i}{\bar{l}} \right\rceil \quad (5)$$

If this invariant is violated, the lower bound of p has not been updated correctly, but other values in the domain of p might satisfy the condition, so the invariant checker returns *MISSING_PROPAGATION*. In a similar way we can derive

$$\forall 0 \leq t < p : \bar{l} < \sum_{\{i \mid \bar{s}_i \leq t < \bar{s}_i + d_i\}} r_i \Rightarrow \text{INCONSISTENT} \quad (6)$$

$$\forall 0 \leq t < p : \underline{l} < \sum_{\{i \mid \bar{s}_i \leq t < \bar{s}_i + d_i\}} r_i \Rightarrow \text{MISSING_PROPAGATION} \quad (7)$$

We are cumulating the resource use over all obligatory parts, i.e. time periods where we know that a task will be active.

The weakest value is *INTERESTING*, which can be used to mark snapshots where a constraint detects a special condition that the user is interested in. We will show its use in section 5.3 to mark nodes where no propagation of a global method was possible. One key advantage of an invariant checker inside a system independent tool is that the invariant code can be shared for all constraint systems that implement a given constraint. As it is written independently from any specific propagation methods, it avoids problems when reused, buggy code in the validation precludes detection of an error. Finally, the invariant checks enhance chances to detect subtle differences in the declarative meaning of a global constraint between systems.

5 Implementation

In this section we discuss the platforms which currently have been integrated with the CP-VIZ tool set, and note some details of the effort required to connect a new system to the visualizer.

5.1 ECLiPSe

We have linked the finite domain *ic* library of the Prolog based ECLiPSe system to CP-VIZ as part of the ECLiPSe ELearning course development. The main requirement was to display sufficient information of the constraint propagation to the students, without overwhelming them with unwanted detail. As custom search routines are very easy to write in ECLiPSe, we also needed an interface which could visualize such routines with minimal overhead. The current interface does not require hooks in the predefined search routines or constraint implementations, but rather uses logic programming features to express the visualization as annotations of the user programs. Visualizers for some 15 global constraints have been implemented so far, together with a series of example programs, based on the course material.

5.2 SICStus

We are currently extending the `clpfd` library module of SICStus Prolog with exported predicates producing XML files for CP-VIZ. The work is being carried out as a part of the interface code for ECLiPSe. Like for ECLiPSe, we do not use any special hooks of SICStus Prolog or its `clpfd` library module and we rely on user program annotations. The implementation replaces the normal `labeling/2` procedure, which takes a list of domain variables, by another procedure taking a list of domain variables annotated with information for display purposes, e.g. variable name. ECLiPSe and SICStus provide different libraries of global constraints, and so the main implementation effort lies in implementing visualizers for the global constraints not provided by ECLiPSe. In particular, we plan to implement 2D and 3D visualizers for the generic multi-dimensional *geost* constraint [2]. The XML files are currently being written with standard Prolog I/O predicates. For efficiency, this is likely to be replaced later by specific XML I/O code.

5.3 Visualization of the Global Constraint SOFTPREC

As a case study for visualizing individual global constraints, consider the visualization of the SOFTPREC constraint arising in the context of the feature subscription problem for telecommunication services. A feature subscription problem is a configuration problem defined by a set of possible features, a set of hard precedence constraints, a set of soft precedence constraints, and a function that maps each feature and each soft precedence constraint to a non-zero integer weight. The objective is to maximize the value of the subscription, which is defined to be the sum of the weights of the features and soft precedences that are included. The soft global precedence constraint SOFTPREC is proposed for solving the feature subscription problem in [11] and [10]. It holds if and only if there is a strict partial order on the selected features subject to the relevant

hard precedence constraints and the selected soft (user) precedence constraints, and the value of the subscription is within the provided bounds.

The algorithms for the pruning rules of SOFTPREC have been implemented in Choco (<http://www.emn.fr/z-info/choco-solver>), which is a Java library for constraint programming. In order to visualize the search tree and the propagation carried out at each node of the search tree, the implementation of SOFTPREC was extended in order to generate and save the trace in the CP-VIZ format. The implementation of this extension was fairly simple. We had to extend two classes of Choco and override some of their methods. While generating the required data for visualization is quite easy, deciding what to visualize and how to visualize it required several iterations.

A distinct advantage of the visualizer is that it is easy to get a sense of the solutions. Visualizing solutions can give more insight than just knowing the numerical value of the solution. It can help a user in deciding whether a given solution with the optimal value is really optimal for him/her or not. The arguments of SOFTPREC that an end-user might be interested in visualizing are the states of the variables associated with optional features, soft precedences, and the value of the subscription being computed. Some of the interesting states are whether a feature (or a user precedence) is included, excluded or undecided, or whether the current state is a result of the last choice or the previous choices.

Figure 7 depicts the search tree (on the left, generated by a branch and bound search algorithm) explored until the node number 38 and the states of the variables (after constraint propagation) at that node, when solving an instance of feature subscription with 20 features and 10 user precedences. In Figure 7 leaf-nodes of the search tree corresponding to feasible solutions are shown in green (light gray), dead-ends are shown in red (dark gray), and the current node (node number 38) is shown in blue (larger size). Figure 7 (right) visualizes the state of the variables after reaching the fix point completing the propagation at node number 38.

The states of the features are visualized using a vector of cells (shown at the bottom and to the right). When a feature is undecided the corresponding cell is unlabeled. If a feature is included or excluded then the cell is labeled with either 1 or 0. The difference between the features that are included/excluded in the current node from those that are decided in the earlier nodes is made through the difference in the background color of the cells. The states of the variables associated with the soft precedence constraints are visualized through a matrix of cells. Each soft precedence constraint $i \prec j$ is associated with a cell in row i and column j . If a soft precedence $i \prec j$ holds then the corresponding cell is labeled with 1 and if it is violated then the corresponding cell is labeled with 0, and undecided soft precedence is labeled with 01. The bounds of the value of the subscription being computed is displayed on the left bottom of Figure 7 (right).

SOFTPREC internally maintains transitivity on the hard precedence constraints. A hard precedence constraint, $i \prec j$, means that if features i and j are included then i must precede j . From a developer's point of view it is interesting to visualize the states of these variables, which is done through the background colors of the cells in the matrix. SOFTPREC also elicits and maintains incompatibilities between undecided features through the states of these variables. An incompatibility between undecided features i and j is visualized by placing a box around the cell in row i and column j . For example,

in Figure 7 (right) the cell in the second row and fifth column is surrounded by a red box which denotes that feature 2 and feature 5 are incompatible. This helps in seeing patterns in the incompatibilities between pairs of features. Within SOFTPREC bounds are computed by associating a graph with a set of incompatibilities, and computing the violation cost of each component of the graph. The components are also visualized by using different colors for the incompatibilities of different components. When it comes to describing the pruning rules of SOFTPREC it is much easier to explain them through visualization. Initially it was agreed to implement a static variable ordering for SOFTPREC that chooses variables associated with soft precedences before the variables associated with features. However, after visualizing the search tree, we discovered that the intended variable ordering was not implemented in the right way. Another advantage of visualization is that it can help in understanding the impact of the strength of different pruning rules.

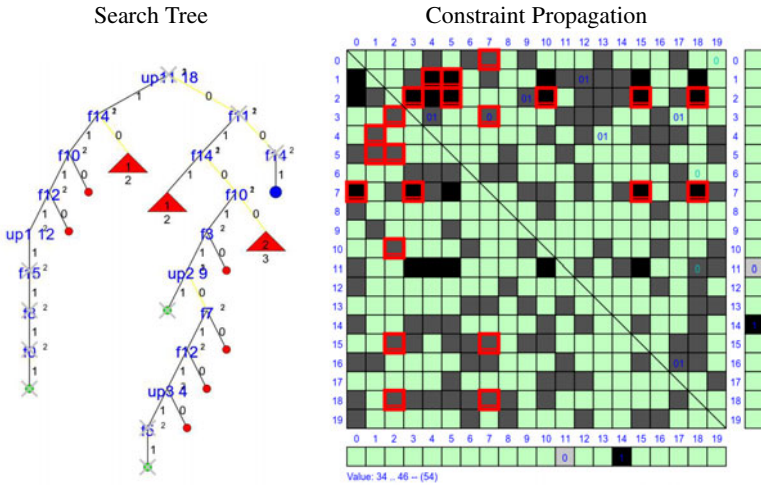


Fig. 7. Example of SOFTPREC Global Constraint Visualization

5.4 JSR331

The Java Specification Request (JSR) 331 (<http://jcp.org/en/jsr/detail?id=331>) is a working group in the Java Community Process trying to propose an open, standard constraint programming API for Java. As part of a reference implementation we have considered the use of CP-VIZ as an example of a visualization extension for the standard API. Figure 8 shows a code example for an annotated N-queens program in the proposed standard syntax. For most classes (Problem, Solver, individual global constraints), variants which incorporate the visualization capabilities of CP-VIZ are provided. By creating for example a new constraint from ALLDIFFERENTVISUAL instead of ALLDIFFERENT, a visualization for this constraint will be provided. Note that not all constraints and variables need to be annotated, the user can concentrate on only parts of the model, if required. Figure 9 shows an UML sequence diagram for the

```

public class QueensVisual {
    public static void main(String [] args) {
        ProblemVisual problem = new ProblemVisual("Queens");
        int size = 16;
        problem.startVisualization("QueensProblem.log");
        Var[] x = problem.varArray("x",0, size-1, size);
        Var[] x1 = new Var[size];
        Var[] x2 = new Var[size];
        for (int i = 0; i < size; i++) {
            x1[i] = x[i].add(i);
            x2[i] = x[i].sub(i);
        }
        problem.register(x);
        new AllDifferentVisual(x).post();
        problem.snapshot();
        new AllDifferent(x1).post();
        new AllDifferent(x2).post();
        SolverVisual solver = new SolverVisual(problem);
        solver.startVisualization("QueensSolver.log");
        Solution solution = solver.findSolution();
        solver.stopVisualization();
        problem.stopVisualization();
    }
}

```

Fig. 8. JSR 331 Example: Visualization of N-Queens Problem

interaction of the Application, VisualProblem, VisualSolver and Constraint classes which shows how the JSR331 implementation builds on the interface of Figure 5.

6 Future Work and Conclusions

While the current CP-VIZ system already provides many useful features for understanding and improving constraint programs, there are a number of features that would improve its capabilities:

- At the moment the system can tell the user which choice led to a failure, but can not provide a more detailed explanation. It would be helpful if we can integrate some explanation tools which can provide automatically derived explanations of failures.
- Much of the development time for a constraint application is taken up with comparing different possible design choices. We will study how to best compare search trees and constraint and variable visualizations from multiple runs in a single display.
- The invariant checker provides a useful paradigm for concentrating effort on interesting parts of the search effort, but at the moment the checks are compiled as part of the tool itself. It might be interesting to allow users to specify checks interactively, and display such search results inside the visualization.

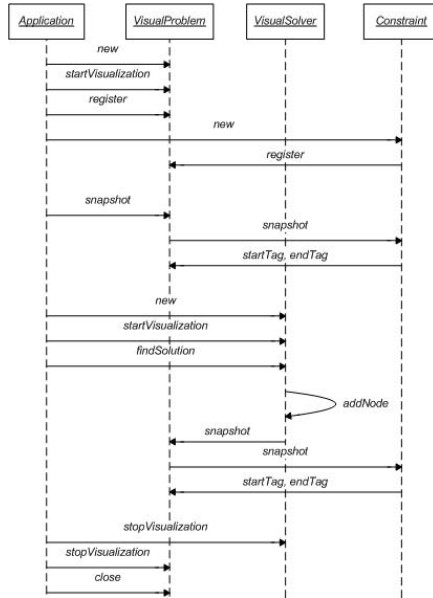


Fig. 9. UML Sequence Diagram - Message Flow between Application and Visualization Classes

By providing an open-source, system independent visualization platform, CP-VIZ can help to reduce the amount of duplicated and redundant work required by system developers, while allowing specific, new features to be added without too much effort. The current documentation and software for CP-VIZ can be found at <http://4c.ucc.ie/~hsimonis/CPVIZ/index.htm>.

References

1. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling problems. *Journal of Mathematical and Computer Modelling* 17(7), 57–73 (1993)
2. Beldiceanu, N., Carlsson, M., Poder, E., Sadek, R., Truchet, C.: A generic geometrical constraint kernel in space and time for handling polymorphic k -dimensional objects. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 180–194. Springer, Heidelberg (2007)
3. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog. Technical Report T2005:08, SICS (May 2005)
4. Deransart, P.: Main results of the OADymPPaC project. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 456–457. Springer, Heidelberg (2004)
5. Deransart, P., Hermenegildo, M.V., Mafuszyński, J.: DiSCiPl 1999. LNCS, vol. 1870. Springer, Heidelberg (2000)
6. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., Berthier, F.: The constraint logic programming language CHIP. In: FGCS, pp. 693–702 (1988)
7. Doms, G., Hentenryck, P.V., Michel, L.: Model-driven visualizations of constraint-based local search. *Constraints* 14(3), 294–324 (2009)

8. Epstein, S.L., Li, X.: Cluster graphs as abstractions for constraint satisfaction problems. In: Bulitko, V., Beck, J.C. (eds.) SARA. AAAI, Menlo Park (2009)
9. Hulubei, T.: Refutation Analysis for Constraint Satisfaction Problems. PhD thesis, University College Cork (2007)
10. Lesaint, D., Mehta, D., O'Sullivan, B., Quesada, L., Wilson, N.: A Soft Global Precedence Constraint. In: IJCAI 2009, Pasadena, CA, USA (2009)
11. Lesaint, D., Mehta, D., O'Sullivan, B., Quesada, L., Wilson, N.: Consistency techniques for finding an optimal relaxation of a feature subscription. In: Proceeding of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008), pp. 283–290 (2008)
12. Meier, M.: Debugging constraint programs. In: Montanari, U., Rossi, F. (eds.) CP 1995. LNCS, vol. 976, pp. 204–221. Springer, Heidelberg (1995)
13. Nieuwenhuis, R.: A cumulative scheduling problem. Personal Communication (2008)
14. Schulte, C.: Oz Explorer: A visual constraint programming tool. In: ICLP, Leuven, Belgium, pp. 286–300 (1997)
15. Simonis, H.: An ECLiPSe ELearning course (2009), <http://4c.ucc.ie/~hsimonis/ELearning/index.htm>
16. Simonis, H., Aggoun, A.: Search-tree visualisation. In: Deransart et al [5], pp. 191–208.
17. Simonis, H., Aggoun, A., Beldiceanu, N., Bourreau, E.: Complex constraint abstraction: Global constraint visualisation. In: Deransart et al [5], 299–317

Database Constraints and Homomorphism Dualities^{*}

Balder ten Cate¹, Phokion G. Kolaitis^{1,2}, and Wang-Chiew Tan^{1,2}

¹ University of California Santa Cruz

² IBM Research-Almaden

Abstract. Global-as-view (GAV) constraints form a class of database constraints that has been widely used in the study of data exchange and data integration. Specifically, relationships between different database schemas are commonly described by a schema mapping consisting of a finite set of GAV constraints. Such schema mappings can be viewed as representations of an infinite set of data examples. We study the following problem: when is finite set of GAV constraints uniquely characterizable via a finite set of data examples? By establishing a tight connection between this problem and homomorphism dualities, we obtain a simple criterion for unique characterizability. We also pinpoint the computational complexity of the corresponding decision problem.

1 Introduction and Summary of Results

Since the early days of the relational data model, constraints have played a major role in both the theory and the practice of database systems. In the 1970s and the 1980s, several different types of database constraints, also known as *database dependencies*, were introduced and studied; these include functional dependencies, inclusion dependencies, multi-valued dependencies, and several other classes of dependencies that were used to capture a variety of semantic restrictions that the allowable data must satisfy (see [1] for a survey). In recent years, database dependencies have been used to formalize and study different facets of information integration, which is the problem of accessing and processing data residing in multiple heterogeneous sources. Two prominent facets of information integration are data exchange and data integration (see the surveys [2] and [3]). A key role in the formalization of both data exchange and data integration, as well as of other information integration tasks, is played by the notion of a *schema mapping*. Intuitively, a schema mapping is a specification that describes the relationships between two database schemas, a *source schema* and a *target schema*. More precisely, a schema mapping is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ with \mathbf{S} a source schema, \mathbf{T} a target schema disjoint from \mathbf{S} , and Σ a finite set of constraints involving the schemas \mathbf{S} and \mathbf{T} . The constraints in Σ are typically expressed as formulas of a logical formalism. In particular, the class of *source-to-target tuple-generating dependencies* (in short, *s-t tgds*) is the most extensively studied and widely used collection of schema mapping constraints to date, as it strikes a good balance between expressive power and desirable algorithmic properties. By definition, an s-t tgd is a first-order formula of the form

$$\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y})),$$

^{*} Ten Cate, Kolaitis, and Tan are supported by NSF grant IIS-0430994 and NSF grant IIS-0905276. Tan is also supported by a NSF CAREER award IIS-0347065.

where $\varphi(\mathbf{x})$ is a conjunction of atoms over \mathbf{S} , each variable in \mathbf{x} occurs in at least one atom in $\varphi(\mathbf{x})$, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms over \mathbf{T} with variables in \mathbf{x} and \mathbf{y} (here, an *atom* is a formula $P(x_1, \dots, x_m)$, where P is a relation symbol and x_1, \dots, x_m are variables, not necessarily distinct). Intuitively, an s-t tgd asserts that whenever a certain “pattern” is realized in the source, then another “pattern” must be realized in the target. Schema mappings specified by s-t tgds contain as important special cases the class of *global-as-view* (GAV) schema mappings and the class of *local-as-view* (LAV) schema mappings; both GAV and LAV schema mappings are widely used and are supported by many information integration tools. A GAV schema mapping is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that every constraint in Σ is an s-t tgd in which the right-hand side consists of a single atom, that is, it has the form

$$\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow T(\mathbf{x})),$$

where $T(\mathbf{x})$ is an atom over the target schema. Intuitively, a GAV constraint specifies that a target relation is described in terms of certain source relations. A LAV schema mapping is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that every constraint in Σ is an s-t tgd in which the left-hand side consists of a single atom, that is, it has the form

$$\forall \mathbf{x}(S(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y})),$$

where $S(\mathbf{x})$ is an atom over the source schema. Intuitively, a LAV constraint specifies that a source relation is described in terms of certain target relations. For example, suppose that we wish to form a target relation by deleting the last column from a ternary source relation. This is captured by an s-t tgd of the form $\forall x, y, z(S(x, y, z) \rightarrow U(x, y))$; note that this is both a GAV and a LAV constraint. Similarly, suppose that we wish to form a target relation by appending a column to some binary source relation. This is captured by an s-t tgd of the form $\forall x, y(R(x, y) \rightarrow \exists zT(x, y, z))$, which is a LAV constraint, but not a GAV constraint. Finally, suppose that we wish to form a target relation by joining two binary source relations along the second column of the first relation and the first column of the second. This is captured by an s-t tgd of the form $\forall x, y, z(P(x, y) \wedge R(y, z) \rightarrow T(x, y, z))$, which is a GAV constraint, but not a LAV constraint.

Background on Schema Mappings and Data Examples. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping in which Σ is a finite set of s-t tgds. A *data example* is a pair (I, J) such that I is a source database and J is a target database. If a data example (I, J) satisfies every s-t tgd in Σ , then we say that J is a *solution for I w.r.t. \mathcal{M}* . Every schema mapping \mathcal{M} gives rise to the following *data exchange* problem: given a source database I , construct a solution J for I w.r.t. \mathcal{M} . In general, a source database I may have an infinite number of solutions. This raises the question: which solution J for I should we chose to materialize in solving the data exchange problem? This question was addressed in [4], where the notion of a *universal solution* was introduced. By definition, a universal solution J for I w.r.t. a schema mapping \mathcal{M} is a solution J for I such that for every solution K for I w.r.t. \mathcal{M} , there is a (not necessarily surjective) homomorphism $h : J \rightarrow K$ that is constant on every element of J occurring in I . Intuitively, a universal solution for I is a “most general” solution for I ; moreover, a

universal solution represents, in a precise technical sense, the entire space of solutions for I . Finally, as shown in [4], given a source database I , a *canonical* universal solution for I can be constructed in time bounded by a polynomial in the size of I using the *chase procedure*. By now, universal solutions have become the standard semantics in data exchange (see [5] for a recent survey).

A schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a finite set of s-t tgds, is a syntactic object that provides a finite representation for the infinite space

$$\{(I, J) : (I, J) \text{ is a data example and } J \text{ is a solution for } I \text{ w.r.t. } \mathcal{M}\}.$$

In [6], the following problem was investigated: can this infinite space of data examples be “captured” by a finite set of data examples? The motivation for this problem is that, since schema mappings arising in real-life applications can be quite complex, one would like to use “good” data examples that illustrate the schema mapping at hand and aid in its understanding and refinement. The problem of “capturing” a schema mapping by finitely many data examples was formalized by introducing the notion of *unique characterizability* of a schema mapping via a finite set of data examples of a certain type w.r.t. to a class of s-t tgds. The main focus of [6] was on *universal examples*, where a universal example for \mathcal{M} is a data example (I, J) such that J is a universal solution for I w.r.t. \mathcal{M} . In this case, the concept of unique characterizability takes the following precise form. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping, let \mathcal{C} be a class of s-t tgds such that $\Sigma \subseteq \mathcal{C}$, and let \mathcal{U} be a finite set of universal examples for \mathcal{M} . We say that \mathcal{M} is *uniquely characterized by \mathcal{U} w.r.t. to \mathcal{C}* if, whenever $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ is a schema mapping such that $\Sigma' \subseteq \mathcal{C}$ and every data example in \mathcal{U} is a universal example for \mathcal{M}' , then Σ' is logically equivalent to Σ . In other words, up to logical equivalence, \mathcal{M} is the only schema mapping with s-t tgds from \mathcal{C} for which \mathcal{U} is a set of universal examples. One of the main results in [6] is that every LAV schema mapping is uniquely characterized by a finite set of universal examples w.r.t. the class of all LAV constraints. On the other hand, it is also shown in [6] that there are natural GAV schema mappings that cannot be uniquely characterized by any finite set of universal examples w.r.t. to the class of GAV schema mappings. It should be noted that the proof of this result made use of a generalization ([7, Theorem 3.15]) of Erdős’ well known theorem asserting the existence of graphs of arbitrary large girth and chromatic number.

Summary of Results. Our aim in this paper is to address the following questions. Which GAV schema mappings are uniquely characterizable by a finite set of universal examples w.r.t. the class of all GAV schema mappings? Is there an algorithm to tell whether or not a given GAV schema mapping is uniquely characterizable by a finite set of universal examples w.r.t. the class of all GAV schema mappings? If so, what is the exact complexity of this problem? For simplicity, from now on, the term “uniquely characterizable” will mean uniquely characterizable by a finite set of universal examples w.r.t. the class of all GAV schema mappings. Our first main result yields a necessary and sufficient condition for a GAV schema mapping to be uniquely characterizable. This criterion unveils a tight (and rather unexpected) connection between unique characterizability and homomorphism dualities, which we describe in what follows.

Informally, a homomorphism duality is an equivalence between the existence of a homomorphism *to* a structure and the non-existence of a homomorphism *from* the same

Source schema: {Manages}; Target schema: {CEO, TopManager}

GAV constraints:

$$(\sigma_1) \quad \forall x (\text{Manages}(x, x) \rightarrow \text{CEO}(x))$$

$$(\sigma_2) \quad \forall x, y, z (\text{Manages}(x, x) \wedge \text{Manages}(x, y) \wedge \text{Manages}(y, z) \rightarrow \text{TopManager}(y))$$

Fig. 1. Example of a GAV schema mapping

structure. The prototypical example of a homomorphism duality is the well-known characterization of 2-colorability: for every graph G , there is a homomorphism from some odd cycle C_{2k+1} to G if and only if there is no homomorphism from G to the complete graph K_2 with two nodes. Let \rightarrow be the existence-of-a-homomorphism relation between structures over the same schema, i.e., $B \rightarrow C$ means that there is a homomorphism from B to C . Assume that \mathcal{F} and \mathcal{D} are two collections of structures over the same schema. Following [8], we say that the pair $(\mathcal{F}; \mathcal{D})$ is a *homomorphism duality* if for every structure A , there exists a structure $F \in \mathcal{F}$ such that $F \rightarrow A$ if and only if there is no structure $D \in \mathcal{D}$ such that $A \rightarrow D$; in symbols, $\bigcup_{F \in \mathcal{F}} (F \rightarrow) = \bigcap_{D \in \mathcal{D}} (\not\rightarrow D)$. If $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality, then we say that \mathcal{F} is an *obstruction set for \mathcal{D}* . Thus, the aforementioned characterization of 2-colorability is equivalent to the assertion that the pair $(\{C_{2k+1} : k \geq 1\}; \{K_2\})$ is a homomorphism duality; moreover, $\{C_{2k+1} : k \geq 1\}$ is an obstruction set for $\{K_2\}$.

For every GAV schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and every relation symbol T in \mathbf{T} , we construct a finite set $\mathcal{F}_{\mathcal{M}, T}$ of relational structures over a signature consisting of the relations from the source schema \mathbf{S} plus finitely many constant symbols, and show that \mathcal{M} is uniquely characterizable if and only if each $\mathcal{F}_{\mathcal{M}, T}$ is an obstruction set for some finite set $\mathcal{D}_{\mathcal{M}, T}$ of structures. This result provides the aforementioned necessary and sufficient condition for unique characterizability of GAV schema mappings; however, it does not yield immediately an algorithm for testing whether or not a given schema mapping is uniquely characterizable. In [8], it was shown that a finite set \mathcal{F} of homomorphically incomparable core structures is an obstruction set for some finite set \mathcal{D} if and only if every structure in \mathcal{F} obeys a certain *acyclicity* condition. This result holds for structures over a signature with relation symbols but no constant symbols. Here, we show that this characterization can be extended to structures over a signature with both relation symbols and constant symbols. In particular, we show that a GAV schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is uniquely characterizable if and only if every structure in the aforementioned finite sets $\mathcal{F}_{\mathcal{M}, T}$ obeys a weaker acyclicity condition, which we call *c-acyclicity*. Moreover, there is an algorithm that, given such a GAV schema mapping, computes a uniquely characterizing set of examples. Informally, the c-acyclicity condition allows for cycles, but every cycle must contain (the interpretation of) a constant symbol. This gives rise to a powerful tool for determining whether or not a given GAV schema mapping is uniquely characterizable. As an illustration of the power of this tool, it follows immediately that the GAV schema mapping \mathcal{M} specified by the single s-t tgd $\forall x, y, z (E(x, z) \wedge E(z, y) \rightarrow P(x, y))$ is uniquely characterizable. In contrast, the GAV schema mapping \mathcal{M}' specified by the single s-t tgd $\forall x, y, z, w (E(x, z) \wedge E(z, y) \wedge E(w, w) \rightarrow P(x, y))$ is not uniquely characterizable.

Finally, from a computational-complexity standpoint, we show that the following problem is NP-complete: given a GAV schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, is it uniquely characterizable? We also show that the computational complexity of this problem drops down to LOGSPACE, if \mathcal{M} is in a certain normalized form in which the left-hand sides of the s-t tgds in Σ are cores. In addition, we obtain results concerning the decidability and computational complexity of several other natural algorithmic problems involving GAV schema mappings, universal examples, and unique characterizability.

Most proofs in this paper are omitted for lack of space.

2 Basic Concepts and Preliminaries

Signatures, Structures, Schemas and Databases. In logic, a *signature* is a collection of relation symbols, function symbols, and constant symbols. Here, we will be concerned only with signatures consisting of finitely many relation symbols R_1, \dots, R_n of designated arities and finitely many constant symbols c_1, \dots, c_k . A *structure* A over such a signature is a tuple $A = (D, R_1^A, \dots, R_n^A, c_1^A, \dots, c_k^A)$, where D is a set, called the *domain* of A , each R_i^A is a relation on D whose arity matches the arity of the relation symbol R_i , and each c_j^A is an element of D . If no constant symbols are present, then we talk about a *relational signature* and about *relational structures* over that signature. In what follows, we will assume that all structures considered are finite, that is, the domain and the relations of the structure are finite. For simplicity of notation and when the structure A at hand is understood from the context, we will often use R_i to denote both the relation symbol R_i and the relation R_i^A interpreting it on A .

In databases, a *schema* is a finite collection of relation symbols R_1, \dots, R_n of designated arities, i.e., a schema is a relational signature. A *database* I over such a schema is a tuple $I = (R_1^I, \dots, R_n^I)$ of finite relations over some domain. Every database I can be identified with a relational structure $(\text{adom}(I), R_1^I, \dots, R_n^I)$, where $\text{adom}(I)$ is the *active domain* of I , that is, the set of all values occurring in the relations of I . In what follows, we will use relational structures and databases in an interchangeable way.

A *homomorphism* from A to B is a function h from the domain of A to the domain of B such that for every relation symbol R_i and every constant symbol c_j : (1) if $(a_1, \dots, a_m) \in R_i^A$, then $(h(a_1), \dots, h(a_m)) \in R_i^B$; and (2) $h(c_j^A) = c_j^B$.

Schema Mappings and Universal Solutions. A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where \mathbf{S} and \mathbf{T} are disjoint schemas, called the *source* schema and the *target* schema, and Σ is a set of source-to-target tuple generating dependencies, as defined in Section 1. If Σ consists entirely of GAV constraints, then \mathcal{M} is called a GAV schema mapping; if Σ consists entirely of LAV constraints, then \mathcal{M} is a LAV schema mapping. In this paper, our main focus will be on GAV schema mappings.

Figure 1 contains an example of a GAV schema mapping; it will be our running example throughout paper. In this example, the source database contains information about managerial relationships in a company using a binary relation “Manages”, while the target database contains information about managerial roles, using the unary relations “CEO” and “TopManager”. Incidentally, note that σ_1 is both a GAV constraint and a LAV constraint, while σ_2 is a GAV constraint but not a LAV constraint.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping. A *data example* is a pair (I, J) , where I is a source database (i.e., a database over \mathbf{S}) and J is a target structure (i.e., a database over \mathbf{T}). If I is a source database, then a *solution for I w.r.t. \mathcal{M}* is a target database J such that the data example (I, J) satisfies every constraint in Σ . A *universal solution for I w.r.t. \mathcal{M}* is a solution J for I w.r.t. \mathcal{M} such that for every solution J' for I w.r.t. \mathcal{M} , there is a homomorphism $h : J \rightarrow J'$ such that h is constant on the active domain $\text{adom}(I)$ of I , i.e., $h(a) = a$, for each $a \in \text{adom}(I) \cap \text{dom}(h)$. It was shown in [4] that if \mathcal{M} is any schema mapping specified by s-t tgds, then every source structure I has a *canonical* universal solution $\text{CanSol}_{\mathcal{M}}(I)$, which can be constructed in time bounded by a polynomial in the size of I . If \mathcal{M} is a GAV schema mapping and I is a source database, then the active domain of $\text{CanSol}_{\mathcal{M}}(I)$ is contained in $\text{adom}(I)$. In fact, in the case of GAV schema mappings, $\text{CanSol}_{\mathcal{M}}(I)$ is the only universal solution for I such that its active domain is contained in $\text{adom}(I)$. Moreover, the relations of $\text{CanSol}(I)$ consist precisely of all tuples that are “dictated” by the GAV constraints of \mathcal{M} , in the sense that they are the right-hand-side of a GAV constraint σ of \mathcal{M} , under a variable assignment that makes the left-hand-side of σ true in I . Note, however, that the state of affairs is more complicated for non-GAV schema mappings (and, in particular, for LAV schema mappings), since the active domain of $\text{CanSol}(I)$ may contain values not occurring in $\text{adom}(I)$.

Characterizing Schema Mappings via Data Examples. In [6], different notions of data examples were considered for “illustrating” the semantics of a schema mapping, such as positive examples, negative examples, and universal examples.

- A *positive example* (I, J) for a schema mapping \mathcal{M} is a data example for \mathcal{M} such that J is a solution for I w.r.t. \mathcal{M} .
- A *negative example* (I, J) for a schema mapping \mathcal{M} is a data example for \mathcal{M} such that J is *not* a solution for I w.r.t. \mathcal{M} .
- A *universal example* (I, J) for a schema mapping \mathcal{M} is a data example for \mathcal{M} such that J is a universal solution for I w.r.t. \mathcal{M} .

Among these, *universal examples* were shown in [6] to be the most promising type of data example for capturing the semantics of a schema mapping. Specifically, the central question studied in [6] is the *unique characterizability* problem: can a schema mapping be “captured” by a finite set of data examples of particular types w.r.t. a class of s-t tgds? In the context of universal examples, the unique characterizability problem is defined as follows. First, if \mathcal{M} is a schema mapping \mathcal{M} and \mathcal{U} is a set of data examples, we say that \mathcal{M} *fits the universal examples \mathcal{U}* if all data examples in \mathcal{U} are universal examples of \mathcal{M} . We say that a schema mapping \mathcal{M} is *uniquely characterized by a finite set of universal examples \mathcal{U} w.r.t. a class of s-t tgds \mathcal{C}* if \mathcal{M} fits \mathcal{U} and for every finite set $\Sigma' \subseteq \mathcal{C}$ such that $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ fits \mathcal{U} , we have that Σ and Σ' are logically equivalent. Similar definitions apply in the case of positive examples and negative examples.

The following results were established in [6]. First, there are LAV schema mappings that are not uniquely characterizable by any finite set of positive and negative examples w.r.t. to the class of all LAV constraints. In contrast, every LAV schema mapping is uniquely characterized by some finite set of universal examples w.r.t. the class of all LAV constraints. Moreover, this positive result extends to the much broader classes of *n-modular* schema mappings [9], where n is a positive integer. The state of affairs

for GAV schema mappings and universal examples turned out to be quite different, as revealed by the next result.

Theorem 1. ([6]) *The following statements are true.*

- *The schema mapping specified by the GAV constraint $\forall x, y(S(x, y) \rightarrow T(x, y))$ is uniquely characterizable w.r.t. the class of GAV constraints by the universal examples given in Figure 2*
- *The schema mapping specified by the GAV constraint $\forall x, y, z(S(x, y) \wedge R(z, z) \rightarrow T(x, y))$ is not uniquely characterizable by any finite set of universal examples w.r.t. the class of GAV constraints.*

The constraint in the first part of Theorem 1 can be thought of as a “copy” constraint that copies the relation S into the relation T . The constraint in the second part of Theorem 1 can be thought of as a “copy constraint with a trigger”: S is copied into T , provided the relation R contains a self-loop. What is the reason that these two GAV constraints have such different properties? More generally, which GAV schema mappings are uniquely characterizable via universal examples w.r.t. to the class of all GAV constraints? Is the associated decision problem (whether or not a given GAV schema mapping is uniquely characterizable) decidable?

Before embarking on the study of these questions, we point out that unique characterizability of GAV schema mappings via universal examples w.r.t. the class of all GAV constraints is equivalent to unique characterizability via positive and negative examples.

Proposition 1. *For GAV schema mappings \mathcal{M} , the following are equivalent w.r.t. the class of all GAV constraints:*

1. \mathcal{M} is uniquely characterizable by positive and negative examples,
2. \mathcal{M} is uniquely characterizable by universal examples,
3. \mathcal{M} is uniquely characterizable by positive, negative, and universal examples

Proposition 1 shows that, in the GAV setting, unique characterizability is a particularly robust notion, in the sense that it does not depend on whether we consider universal examples, or positive and negative examples. As we will focus on GAV schema mappings, we will therefore simply speak of *unique characterizability*, meaning *unique characterizability by a finite set of universal examples w.r.t. the class of GAV constraints*.

3 Homomorphism Dualities and Unique Characterizations

In this section, we establish a connection between unique characterizations of GAV schema mappings on the one hand, and homomorphism dualities for relational structures on the other. Specifically, we show that the problem of testing whether a GAV schema mapping is uniquely characterizable can be reduced to a certain problem concerning the existence of a homomorphism duality; furthermore, the problem of testing

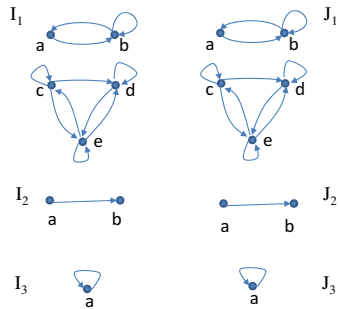


Fig. 2. Universal examples uniquely characterizing the copy constraint $\forall x, y(S(x, y) \rightarrow T(x, y))$

whether a GAV schema mapping is uniquely characterized by a given set of universal examples can be reduced to the question of whether a given pair of sets of structures is a homomorphism duality. Since these two problems concerning homomorphism dualities are decidable (cf. Section 4), we will be able to derive decidability results for the two problems concerning unique characterizations (cf. Section 5).

Homomorphism Dualities. As described in Section 1 a homomorphism duality is an equivalence between the existence of a homomorphism *to* a structure and the non-existence of a homomorphism *from* the same structure. We will work with a finite signature consisting of relation symbols and constant symbols; recall that all structures considered here are assumed to be finite. Given a structure A , we denote by $A \rightarrow$ the set of all structures (over the same signature) that A has a homomorphism into; in symbols, $A \rightarrow = \{B : A \rightarrow B\}$. Similarly, $\not\rightarrow A$ is the set of all structures that do not have a homomorphism into A , i.e., $\not\rightarrow A = \{B : B \not\rightarrow A\}$.

Definition 1. Let \mathcal{F} and \mathcal{D} be two sets of structures. We say that the pair $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality if $\bigcup_{F \in \mathcal{F}} (F \rightarrow) = \bigcap_{D \in \mathcal{D}} (\not\rightarrow D)$. If $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality, then we say that \mathcal{F} is an obstruction set for \mathcal{D} .

If $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality, it means that the class of all structures is partitioned into two disjoint subclasses, namely, the subclass $\bigcup_{F \in \mathcal{F}} (F \rightarrow)$ of those structures that some structure in \mathcal{F} has a homomorphism into, and the subclass $\bigcup_{D \in \mathcal{D}} (\not\rightarrow D)$ of those structures that have a homomorphism into some structure in \mathcal{D} . This is illustrated in Figure 3 (where, intuitively, the direction of homomorphisms is upward).

A homomorphism duality in which both sets of structures are singletons is called a *simple homomorphism duality pair*, and is typically written without curly braces. Homomorphism dualities, and in particular simple homomorphism duality pairs, have been studied extensively in graph theory (where they are used to gain understanding of the structure of the lattice of graphs and homomorphisms, cf. [7]) and in the context of constraint satisfaction problems (where they have been used in order to identify classes of tractable constraint satisfaction problems, cf. e.g., [10]).

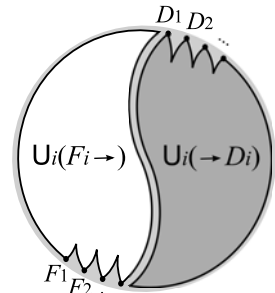


Fig. 3. A homomorphism duality

Homomorphism Dualities and Unique Characterizations. We will now establish the fundamental connection between unique characterizations of GAV schema mappings and homomorphism dualities. In order to state the result, we associate a *canonical structure* with every GAV constraint. Specifically, consider a GAV constraint

$$\sigma = \forall x_1, \dots, x_m (\phi(x_1, \dots, x_m) \rightarrow T(y_1, \dots, y_k))$$

over a source schema $\mathbf{S} = \{S_1, \dots, S_n\}$ and a target schema $\mathbf{T} = \{T, \dots\}$, with $y_1, \dots, y_k \in \{x_1, \dots, x_m\}$. The *canonical structure* associated with σ is the following structure A_σ over the signature $\{S_1, \dots, S_n, c_1, \dots, c_k\}$:

$$A_\sigma = (\{x_1, \dots, x_m\}, S_1^{A_\sigma}, \dots, S_n^{A_\sigma}, c_1^{A_\sigma}, \dots, c_k^{A_\sigma})$$

where each relation $S_i^{A_\sigma}$ consists of the tuples in the atoms of ϕ that involve S_i , and each $c_j^{A_\sigma} = y_j$. In database-theory terms, A_σ is the canonical instance of the left-hand side of σ (viewed as a conjunctive query), expanded with constant symbols marking the exact sequence of exported variables y_1, \dots, y_k . For a GAV schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and a target relation $T \in \mathbf{T}$, we denote by $\mathcal{F}_{\mathcal{M}, T}$ the set of all canonical structures of GAV constraints $\sigma \in \Sigma$ that use the target relation T in their right-hand-side.

Theorem 2. *A GAV schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is uniquely characterizable if and only if for each $T \in \mathbf{T}$, $\mathcal{F}_{\mathcal{M}, T}$ is an obstruction set for a finite set of structures.*

Before we present the proof of Theorem 2 let us illustrate the result by revisiting our running example in Figure 1. The canonical structures A_{σ_1} and A_{σ_2} of the GAV constraints σ_1, σ_2 can be depicted as $\boxed{G \cdot c_1}$ and $\boxed{G \cdot \longrightarrow \cdot c_1 \longrightarrow \cdot}$, respectively, where an arrow indicates that two elements stand in the Manages relation. Since σ_1 and σ_2 use different target relations, Theorem 2 tells us that, in order to determine whether this GAV schema mapping is uniquely characterizable, it is enough to test whether each of these structures, taken as a singleton set, is an obstruction set for a finite set of structures. As it turns out, $\{A_{\sigma_1}\}$ is indeed an obstruction set for a finite set of structures (in fact, the reader may easily verify that $(A_{\sigma_1}; B)$ is a simple homomorphism duality pair, with B the structure depicted by $\boxed{\cdot c_1 \longleftrightarrow \cdot \odot}$). On the other hand, $\{A_{\sigma_2}\}$ is not an obstruction set for any finite set of structures, as will follow from results presented in Section 4. It follows that our example schema mapping is not uniquely characterizable.

We will now proceed with the proof of Theorem 2. We will use the following convenient notation, familiar from logic. If A is a structure over the signature $\{S_1, \dots, S_n\}$, and a_1, \dots, a_k is a sequence of (not necessarily distinct) elements of the domain of A , then we denote by $\langle A, a_1, \dots, a_k \rangle$ the structure over the signature $\{S_1, \dots, S_n, c_1, \dots, c_k\}$ that has the same domain as A and agrees with A on the denotation of the relations S_1, \dots, S_n , and in which each constant symbol c_i denotes the element a_i . In other words, $\langle A, a_1, \dots, a_k \rangle$ is identical to A except that the elements a_1, \dots, a_k are named using fresh constant symbols.

Proof (of Theorem 2). First, we show that, when it comes to the question of unique characterizability, we can restrict attention to schema mappings for a single target relation. This is stated by the next lemma. For any relation $T \in \mathbf{T}$, we denote by $\mathcal{M}|_T$ the schema mapping $(\mathbf{S}, \{T\}, \Sigma')$ where $\Sigma' \subseteq \Sigma$ consists of all GAV constraints whose right-hand side contains the target relation T .

Lemma 1. *\mathcal{M} is uniquely characterizable if and only if for each $T \in \mathbf{T}$, the schema mapping $\mathcal{M}|_T$ is uniquely characterizable.*

We will also make use of the following fact concerning canonical universal solutions of GAV schema mappings (cf. [9]):

Lemma 2. *For all source structures I_1, I_2 , every homomorphism $h : I_1 \rightarrow I_2$ is also a homomorphism $h : \text{CanSol}_{\mathcal{M}}(I_1) \rightarrow \text{CanSol}_{\mathcal{M}}(I_2)$.*

We now proceed with the main proof. By Lemma 1 we may assume $\mathbf{T} = \{T\}$, and show that \mathcal{M} is uniquely characterizable if and only if $\mathcal{F}_{\mathcal{M}, T}$ is an obstruction set for a finite set of structures.

(\Rightarrow) Let \mathcal{U} be a set of universal examples uniquely characterizing \mathcal{M} . Let \mathcal{D} be the set $\{\langle I, \mathbf{a} \rangle \mid (I, J) \in \mathcal{U}, \mathbf{a} \in \text{dom}(I)^k \setminus T^J\}$. We claim that $(\mathcal{F}_{\mathcal{M}, T}; \mathcal{D})$ is a homomorphism duality. To see this, it is enough to observe that, for all source structures I and for all tuples \mathbf{a} , we have that:

- $\langle F, \mathbf{b} \rangle \rightarrow \langle I, \mathbf{a} \rangle$ for some $\langle F, \mathbf{b} \rangle \in \mathcal{F}_{\mathcal{M}, T}$ if and only if $T(\mathbf{a}) \in \text{CanSol}_{\mathcal{M}}(I)$,
- $\langle I, \mathbf{a} \rangle \rightarrow \langle D, \mathbf{b} \rangle$ for some $\langle D, \mathbf{b} \rangle \in \mathcal{D}$ if and only if $T(\mathbf{a}) \notin \text{CanSol}_{\mathcal{M}}(I)$.

The first item follows immediately from the construction of \mathcal{F} . The left-to-right direction of the second item follows from Lemma 2. The right-to-left direction of the second item can be shown by contradiction: suppose $\text{CanSol}_{\mathcal{M}}(I)$ does not contain $T(\mathbf{a})$ and $\langle I, \mathbf{a} \rangle$ does not homomorphically map into any $\langle D, \mathbf{b} \rangle \in \mathcal{D}$. Let \mathcal{M}' extend \mathcal{M} with an extra GAV constraint, namely the canonical GAV constraint of $\langle I, \mathbf{a} \rangle$. Clearly, \mathcal{M}' is not logically equivalent to \mathcal{M} , but it is not hard to see that \mathcal{M}' fits the universal examples \mathcal{U} , contradicting the fact that the universal examples \mathcal{U} uniquely characterize \mathcal{M} .

(\Leftarrow) Let \mathcal{D} be a finite set of structures such that $(\mathcal{F}_{\mathcal{M}, T}; \mathcal{D})$ is a homomorphism duality. Let $\mathcal{U} = \{\langle I, \text{CanSol}_{\mathcal{M}}(I) \rangle \mid \langle I, \mathbf{a} \rangle \in \mathcal{F}_{\mathcal{M}, T} \cup \mathcal{D}\}$. We claim that \mathcal{U} uniquely characterizes \mathcal{M} . For, consider any schema mapping \mathcal{M}' fitting the universal examples in \mathcal{U} , any source structure I and any k -tuple \mathbf{a} of elements from the domain of I , where k is the arity of T . There are two cases:

The first case is where $\langle F, \mathbf{b} \rangle \rightarrow \langle I, \mathbf{a} \rangle$ for some $\langle F, \mathbf{b} \rangle \in \mathcal{F}_{\mathcal{M}, T}$. By construction of $\mathcal{F}_{\mathcal{M}, T}$, we have that $\text{CanSol}_{\mathcal{M}}(F)$, hence also $\text{CanSol}_{\mathcal{M}'}(F)$, contains $T(\mathbf{b})$. It follows by Lemma 2 that $\text{CanSol}_{\mathcal{M}}(I)$ and $\text{CanSol}_{\mathcal{M}'}(I)$ contain $T(\mathbf{a})$.

The second case is where $\langle I, \mathbf{a} \rangle \rightarrow \langle D, \mathbf{b} \rangle$ for some $\langle D, \mathbf{b} \rangle \in \mathcal{D}$. It follows from the duality and from the construction of $\mathcal{F}_{\mathcal{M}, T}$ that $\text{CanSol}_{\mathcal{M}}(D)$, and therefore also $\text{CanSol}_{\mathcal{M}'}(D)$, does not contain $T(\mathbf{b})$. It follows by Lemma 2 that $\text{CanSol}_{\mathcal{M}}(I)$ and $\text{CanSol}_{\mathcal{M}'}(I)$ both do not contain $T(\mathbf{a})$.

This shows that $\text{CanSol}_{\mathcal{M}}(I) = \text{CanSol}_{\mathcal{M}'}(I)$. In other words, \mathcal{M} and \mathcal{M}' are logically equivalent, and hence \mathcal{M} is uniquely characterized by \mathcal{U} . \square

The above result links the notion of unique characterizability to that of being an obstruction set of a finite set of structures. In a similar fashion, we can link unique characterizations themselves to homomorphism dualities. This is expressed by the following Theorem. The proof is a variation of that of Theorem 2.

Theorem 3. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a GAV schema mapping and \mathcal{U} a set of universal examples for \mathcal{M} . For each $T \in \mathbf{T}$, let $\mathcal{F}_T = \{\langle I, \mathbf{a} \rangle \mid (I, J) \in \mathcal{U}, \mathbf{a} \in T^J\}$ and let $\mathcal{D}_T = \{\langle I, \mathbf{a} \rangle \mid (I, J) \in \mathcal{U}, \mathbf{a} \in \text{dom}(I)^k \setminus T^J\}$, where k is the arity of T . Then the following statements are equivalent:*

1. \mathcal{U} uniquely characterizes \mathcal{M} .
2. For each $T \in \mathbf{T}$, $(\mathcal{F}_T; \mathcal{D}_T)$ is a homomorphism duality.

Theorem 2 and 3 reduce questions about unique characterizations to questions about homomorphism dualities. In the remainder of this section, we show that the same applies the other way around (so that we will be able to transfer not only complexity theoretic upper bounds for these questions, but also lower bounds). To state these results, we need a way to associate to each structure a GAV constraint. Given a structure

$$A = (\{a_1, \dots, a_m\}, S_1^A, \dots, S_n^A, c_1^A, \dots, c_k^A)$$

with $c_1^A = a_{i_1}, \dots, c_k^A = a_{i_k}$, we associate with it the *canonical GAV constraint*

$$\sigma_A = \forall x_1, \dots, x_m \left(\bigwedge_{\substack{1 \leq i \leq n \\ (a_{j_1}, \dots, a_{j_\ell}) \in S_i^A}} S_i(x_{j_1}, \dots, x_{j_\ell}) \rightarrow T(x_{i_1}, \dots, x_{i_k}) \right)$$

over the source schema $\mathbf{S} = \{S_1, \dots, S_n\}$ and a target schema $\mathbf{T} = \{T\}$, where T is a k -ary target relation. In other words, the antecedent of σ_A is the atomic diagram of (the purely relational part of) A , while the conclusion of σ_A lists the elements denoted by the constant symbols in order. The reader may verify that, by this definition, the canonical GAV constraint of the canonical structure of a GAV constraint σ is just σ itself (up to renaming of variables and reordering of conjuncts). The *canonical GAV schema mapping* $\mathcal{M}_{\mathcal{F}}$ of a finite set of structures \mathcal{F} is the schema mapping defined by the canonical GAV constraints of the structures in \mathcal{F} .

Theorem 4. *Let \mathcal{F} be a finite set of structures for a signature $\{S_1, \dots, S_n, c_1, \dots, c_k\}$. Then \mathcal{F} is an obstruction set for a finite set of structures if and only if $\mathcal{M}_{\mathcal{F}}$ is uniquely characterizable.*

Theorem 5. *Let \mathcal{F}, \mathcal{D} be finite sets of structures for a signature $\{S_1, \dots, S_n, c_1, \dots, c_k\}$. Let $\mathcal{U}_{\mathcal{F}, \mathcal{D}}$ be the set of all pairs $(I, \text{CanSol}_{\mathcal{M}_{\mathcal{F}}}(I))$ where $I = (D, S_1^A, \dots, S_n^A)$ for some $A = (D, S_1^A, \dots, S_n^A, c_1^A, \dots, c_k^A) \in \mathcal{F} \cup \mathcal{D}$. The following are equivalent:*

1. $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality
2. $\mathcal{M}_{\mathcal{F}}$ is uniquely characterized by $\mathcal{U}_{\mathcal{F}, \mathcal{D}}$, and, moreover, for each structure $A = (D, S_1^A, \dots, S_n^A, c_1^A, \dots, c_k^A) \in \mathcal{D}$, $\text{CanSol}_{\mathcal{M}_{\mathcal{F}}}(A)$ does not contain $T(c_1^A, \dots, c_k^A)$.

4 Results on Homomorphism Dualities

In this section, we will present a characterization of the finite sets of structures \mathcal{F} that are an obstruction set for a finite set of structures \mathcal{D} . For the case of relational signatures without constant symbols, an elegant characterization of such sets \mathcal{F} was established in [8]. Our main contribution in this section is to show that the characterization can be extended in a natural way to structures with constant symbols.

We now introduce some terminology and state two basic facts concerning homomorphism dualities. Recall that, for structures A, B , we write $A \rightarrow B$ if there is a homomorphism from A to B . We say that A and B are *homomorphically equivalent* if $A \rightarrow B$ and $B \rightarrow A$, and we say that A and B are *homomorphically incomparable* if there are neither $A \rightarrow B$, nor $B \rightarrow A$. Every finite structure A is known to have a unique (up to isomorphism) smallest homomorphically equivalent substructure that is homomorphically equivalent to A , which is known as the *core* of A [11]. A structure is said to be a *core* if it is its own core. For a set X of structures, we denote by $\text{core } X$ the set of cores of structures in X , we denote by $\max X$ any subset $Y \subseteq X$ consisting of homomorphically incomparable structures such that for all $A \in X$, there is a $B \in Y$ with $A \rightarrow B$; in a dual manner, we denote by $\min X$ any subset $Y \subseteq X$

consisting of homomorphically incomparable structures such that for all $A \in X$, there is a $B \in Y$ with $B \rightarrow A$. If one reflects on the definition of homomorphism dualities, and keeps in mind Figure 3 the following fact becomes evident (note that, if $A \rightarrow B$, then $(B \rightarrow) \subseteq (A \rightarrow)$ and $(\rightarrow A) \subseteq (\rightarrow B)$):

Fact 6. *Let \mathcal{F} and \mathcal{D} be finite sets of structures. Then $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality if and only if $(\min \text{ core } \mathcal{F}; \max \text{ core } \mathcal{D})$ is a homomorphism duality.*

By construction, $\min \text{ core } \mathcal{F}$ and $\max \text{ core } \mathcal{D}$ have the property that they consist of pairwise homomorphically incomparable core structures. Hence, we may restrict attention to sets \mathcal{F} and \mathcal{D} consisting of pairwise homomorphically incomparable core structures.

The second fact states that for any given finite set of structures \mathcal{F} , there is at most one finite set of structures \mathcal{D} for which \mathcal{F} is an obstruction set, assuming that \mathcal{D} consists of pairwise incomparable core structures. The proof, which we omit, is elementary, using the definition of homomorphism duality and the fact that homomorphisms compose.

Fact 7. *Let $\mathcal{F}, \mathcal{D}, \mathcal{D}'$ be finite sets of homomorphically incomparable core structures such that $(\mathcal{F}; \mathcal{D})$ and $(\mathcal{F}; \mathcal{D}')$ are homomorphism dualities. Then \mathcal{D} and \mathcal{D}' contain the same structures up to isomorphism.*

Known Results for Structures without Constant Symbols. Consider signatures consisting only of relation symbols. The main result from [8] states that a finite set of homomorphically incomparable core structures \mathcal{F} for such a signature is an obstruction set for a finite set of structures \mathcal{D} if and only if each structure from \mathcal{F} obeys a certain acyclicity condition, which we will now define.

A *fact* of a structure A is an expression $R(a_1, \dots, a_m)$ such that R is one of the relations of A and $(a_1, \dots, a_m) \in R$. The *incidence graph* $\text{inc}(A)$ of a structure A is the undirected (bi-partite) graph whose vertices are the elements and the facts of A , and with an edge between an element and a fact if the element occurs in the fact. We call a structure A *acyclic* if $\text{inc}(A)$ is acyclic and no fact of A contains the same element twice. Note that the second condition has to be included explicitly in the definition, since it is not implied by the first (however, in [8], an equivalent definition is given in terms of an incidence multi-graph that may contain several edges between the same fact and element, so that the second condition is not needed).

Theorem 8 ([8]). *Consider a signature consisting of relation symbols only, and let \mathcal{F} be a finite set of homomorphically incomparable core structures. Then \mathcal{F} is an obstruction set for a finite set of structures if and only if every structure in \mathcal{F} is acyclic. Moreover, there is an algorithm that, given such a set \mathcal{F} consisting of acyclic structures, computes a finite set \mathcal{D} such that $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality.*

Incidentally, the algorithm for computing \mathcal{D} from \mathcal{F} given in [8] runs in double exponential time, and no matching lower bound is known (cf. also [12] for improved bounds in the special case of simple homomorphism duality pairs). Foniok et al. [8] also consider the problem of testing whether a given finite set of structures \mathcal{F} has a finite obstruction set. They show (for structures without constant symbols) that this problem is NP-complete and that one can effectively compute an obstruction set if it exists.

A Generalization for the Case with Constant Symbols. In the presence of constant symbols, acyclicity is no longer a necessary condition for being an obstruction set for

a finite set of structures. For instance, consider the structure A depicted by $\boxed{\cdot \longleftrightarrow \cdot}$, and let A' be the expansion of A with a constant symbol c_1 denoting the left-most element, as in $\boxed{c_1 \cdot \longleftrightarrow \cdot}$. Since the incidence graph of A contains a cycle, by Theorem 8 there is no finite set of structures \mathcal{D} such that $(\{A\}; \mathcal{D})$ is a homomorphism duality. The situation for A' is very different. Indeed, if we let B' be the structure depicted by $\boxed{\circ \cdot \longleftrightarrow \cdot \xrightarrow{c_1} \cdot \circ}$, then $(A'; B')$ is a simple homomorphism duality pair.

Nevertheless, Theorem 8 can be extended in a natural way to structures with constant symbols. To this end, we call a structure A over a signature consisting of relation symbols and constant symbols *c-acyclic* if the following both hold:

1. Every cycle in $\text{inc}(A)$ passes through an element named by a constant symbol,
2. If a fact of A contains the same element a twice, a is named by a constant symbol.

Note that for structures without constant symbols, c-acyclicity is equivalent to acyclicity. Also note that the structure A' we discussed above is c-acyclic.

The proof of the following Theorem is based on a reduction to Theorem 8.

Theorem 9. *Consider a signature consisting of relation symbols and constant symbols, and let \mathcal{F} be a finite set of homomorphically incomparable core structures. Then \mathcal{F} is an obstruction set for a finite set of structures if and only if every structure in \mathcal{F} is c-acyclic. Moreover, there is an algorithm that, given such a set \mathcal{F} consisting of c-acyclic structures, computes a finite set \mathcal{D} such that $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality.*

From Theorem 9 we can derive the following computability and complexity results (using the fact that undirected reachability is in LOGSPACE [13]).

Corollary 1. *The following problem is NP-complete: given a finite set of structures \mathcal{F} , determine if \mathcal{F} is an obstruction set for a finite set of structures. The same problem is in LOGSPACE if the input is a set of homomorphically incomparable core structures.*

Corollary 2. *The following problem is decidable: given finite sets of structures \mathcal{F} and \mathcal{D} , determine if $(\mathcal{F}; \mathcal{D})$ is a homomorphism duality.*

5 An Effective Characterization of Unique Characterizability

We now put the results from the previous sections to use. Our main result is an effective characterization of the uniquely characterizable GAV schema mappings.

We say that a GAV schema mapping \mathcal{M} is *normalized* if (i) the canonical structure of the left-hand-side of each GAV constraint is a core, and (ii) for any two GAV constraints for the same target relation, the canonical structures are homomorphically incomparable. Note that every GAV schema mapping is equivalent to a normalized GAV schema mapping, of the same size or smaller, which can be computed in exponential time (in fact, in polynomial time using an NP oracle). For example, consider the schema mapping \mathcal{M} defined by the following GAV constraints:

$$\begin{aligned}
 (\sigma_1) \quad & \forall x, y, z (S(x, y, z) \rightarrow T(x)) \\
 (\sigma_2) \quad & \forall x, y (S(x, y, y) \rightarrow T(x)) \\
 (\sigma_3) \quad & \forall x, y (S(x, x, x) \wedge S(x, y, x) \rightarrow V(x))
 \end{aligned}$$

This schema mapping is not normalized. It violates the first requirement because the canonical structure of σ_3 is not a core, and it violates the second requirement because there is a homomorphism from the canonical structure of σ_1 to the canonical structure of σ_2 . A logically equivalent normalized schema mapping \mathcal{M}' can be obtained from \mathcal{M} by removing the conjunct $S(x, y, x)$ from σ_3 and by removing the entire constraint σ_2 .

We call a GAV schema mapping *c-acyclic* if the canonical structure of each of its GAV constraints is c-acyclic.

Given a GAV constraint σ , a *join cycle* of σ is a sequence $x_1, F_1, x_2, F_2, \dots, x_n, F_n, x_{n+1}$ ($n > 1$) where x_1, x_2, \dots are variables, $x_{n+1} = x_1$, each F_i is an atom from the left-hand-side of σ containing both x_i and x_{i+1} , and $F_i \neq F_{i+1}$ for all $i < n$ (this is to exclude trivial cycles traversing the same edge twice in opposite directions). An *exported variable* of σ is a variable occurring in the right-hand side of σ . Using these two notions, it is easy to see that c-acyclicity is equivalent to saying that the following two conditions hold:

- atoms may not contain two occurrences of the same non-exported variable
- each join cycle passes through an exported variable.

The schema mapping \mathcal{M} described above is not c-acyclic, as the non-exported variable y of σ_2 occurs twice in the same conjunct. However, the normalized schema mapping \mathcal{M}' , where σ_2 is removed and the conjunct $S(x, y, x)$ removed from σ_3 , is c-acyclic. The example schema mapping in Figure 1 is normalized and not c-acyclic.

From Theorem 9, we obtain the following characterization of the uniquely characterizable GAV schema mappings:

Theorem 10. *Every c-acyclic GAV schema mapping is uniquely characterizable, and a uniquely characterizing set of universal examples can be effectively computed from a given c-acyclic GAV schema mapping. Conversely, every uniquely characterizable GAV schema mapping \mathcal{M} is logically equivalent to a c-acyclic GAV schema mapping; in fact, \mathcal{M} is c-acyclic after normalization.*

It follows that, for instance, the schema mapping defined by the GAV constraint $\forall x_1, \dots, x_n (S(x_1, x_2) \wedge \dots \wedge S(x_{n-1}, x_n) \rightarrow T(x_1, x_n))$ is uniquely characterizable, as are schema mappings defined by GAV constraints whose variables are all exported.

In the remainder of this section, we analyze the complexity of various decision problems concerning unique characterizability and unique characterizations. In our complexity analysis, we assume that the source schema and target schema are fixed (and finite). This makes all reductions described in Section 3 polynomial time computable.

Corollary 3. *The following problem is NP-complete: given a GAV schema mapping, is it uniquely characterizable? If the schema mapping is normalized, the problem is in LOGSPACE.*

Corollary 4. *The following problem is decidable: given a GAV schema mapping \mathcal{M} and a finite set of universal examples \mathcal{U} , does \mathcal{U} uniquely characterize \mathcal{M} ?*

Below, we will consider two additional decision problems.

Theorem 11. *The following problem is DP-complete: given a finite set of universal examples \mathcal{U} , is there a GAV schema mapping fitting \mathcal{U} ? If the input consists of ground universal examples, the problem is coNP-complete. In both cases, the hardness holds already for a single universal example.*

Here, by a *ground* data example we mean a data example (I, J) such that the domain of J is a subset of the domain of I . The proof of Theorem 11 in effect, establishes something stronger: from any given finite set of universal examples \mathcal{U} , it is possible to compute in polynomial time a candidate GAV schema mapping $\mathcal{M}_{\mathcal{U}}$, such that if any GAV schema mapping fits the universal examples \mathcal{U} , then $\mathcal{M}_{\mathcal{U}}$ fits (in fact, $\mathcal{M}_{\mathcal{U}}$ is guaranteed to be the *logically weakest fitting schema mapping*, meaning that for any other schema mapping \mathcal{M}' fitting the universal examples \mathcal{U} , the GAV constraints of $\mathcal{M}_{\mathcal{U}}$ logically imply those of \mathcal{M}').

Theorem 12. *The following problem is decidable: given a finite set of universal examples \mathcal{U} , is there a unique schema mapping \mathcal{M} that fits them?*

6 Concluding Remarks

We have established a tight connection between unique characterizability of GAV schema mappings via data examples, and homomorphism dualities, and we used this connection to obtain criteria and complexity results of unique characterizability for GAV schema mappings, and other related results.

The homomorphism dualities we considered in this paper consist of finite sets of structures. In the literature on constraint satisfaction problems, more general types of homomorphism dualities have been studied, for instance where one of the sets consists of infinitely many structures of bounded treewidth [10]. This raises the question whether known results about such dualities can be used to obtain further insights into the unique characterizability of schema mappings.

References

1. Fagin, R., Vardi, M.Y.: The Theory of Data Dependencies - A Survey. In: Proc. of Symposia in Applied Mathematics. Mathematics of Information Processing, vol. 34, pp. 19–71 (1986)
2. Kolaitis, P.G.: Schema Mappings, Data Exchange, and Metadata Management. In: ACM Symposium on Principles of Database Systems (PODS), pp. 61–75 (2005)
3. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: ACM Symposium on Principles of Database Systems (PODS), pp. 233–246 (2002)
4. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. Theoretical Computer Science (TCS) 336(1), 89–124 (2005)
5. Barceló, P.: Logical foundations of relational data exchange. SIGMOD Record 38(1), 49–58 (2009)
6. Alexe, B., Kolaitis, P.G., Tan, W.C.: Characterizing schema mappings via data examples. In: ACM Symposium on Principles of Database Systems, PODS (2010)

7. Hell, P., Nešetřil, J.: *Graphs and Homomorphisms*. Oxford University Press, Oxford (2004)
8. Foniok, J., Nešetřil, J., Tardif, C.: Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *Eur. J. Comb.* 29(4), 881–899 (2008)
9. ten Cate, B., Kolaitis, P.G.: Structural Characterizations of Schema-Mapping Languages. In: *International Conference on Database Theory (ICDT)*, pp. 63–72 (2009)
10. Nešetřil, J., Zhu, X.: On bounded treewidth duality of graphs. *Journal of Graph Theory* 23(2), 151–162 (1998)
11. Hell, P., Nešetřil, J.: The core of a graph. *Discrete Mathematics* 109, 117–126 (1992)
12. Nešetřil, J., Tardif, C.: Short answers to exponentially long questions: Extremal aspects of homomorphism duality. *SIAM J. of Discrete Mathematics* 19(4), 914–920 (2005)
13. Reingold, O.: Undirected st-connectivity in log-space. In: *STOC 2005: Proceedings of the ACM Symposium on Theory of Computing*, pp. 376–385. ACM, New York (2005)

A Box-Consistency Contractor Based on Extremal Functions

Gilles Trombettoni, Yves Papegay, Gilles Chabert, and Odile Pourtallier

COPRIN INRIA, Université Nice–Sophia, LINA Ecole des Mines de Nantes
{Gilles.Trombettoni, Yves.Papegay, Odile.Pourtallier}@sophia.inria.fr,
Gilles.Chabert@emn.fr

Abstract. Interval-based methods can approximate all the real solutions of a system of equations and inequalities. The `Box` interval constraint propagation algorithm enforces *Box consistency*. Its main procedure `BoxNarrow` handles one function f corresponding to the revised constraint, and one variable x , replacing the other variables of f by their current intervals. This paper proposes an improved `BoxNarrow` procedure for narrowing the domain of x when f respects certain conditions. In particular, these conditions are fulfilled when f is polynomial. f is first symbolically rewritten into a new form g . A narrowing step is then run on the non-interval *extremal functions* that enclose the interval function g . The corresponding algorithm is described and validated on several numerical constraint systems.

1 Motivation

Interval-based solvers can solve systems of numerical constraints (i.e., nonlinear equations or inequalities over the reals). Their reliability and increasing performance make them applicable to various domains such as robotics design and kinematics [9], proofs of conjectures [12], robust global optimization [7, 11] and bounded-error parameter estimation [6].

Two main types of *contraction algorithms* allow solvers to filter variable domains, i.e., to reduce the intervals of each variable, without loss of solutions of the system: interval (numerical) analysis methods, like *Interval Newton* [10], and constraint propagation algorithms from constraint programming. The `HC4` and `Box` algorithms [2, 14] are very often used in solving strategies. They perform a propagation loop and filter the variable domains with a specific *revise* procedure (called `HC4-Revise` and `BoxNarrow`) handling the constraints individually. For every pair (c, x) in the system, where c is the numerical constraint $f(x, y_1, \dots, y_{k-1}) = 0$, the `BoxNarrow` contraction procedure is applied to x by considering the uni-variate constraint: $f_{[Y]}(x) = f(x, [y_1], \dots, [y_{k-1}]) = 0$. That is, $f_{[Y]}$ is a function where each variable $y_i \in Y = \{y_1, \dots, y_{k-1}\}$ of f has been replaced by its interval of variation. The important point is that $f_{[Y]}$ is an interval function: to any $x \in \mathbb{R}$, $f_{[Y]}(x)$ is an interval. Thus, the iterative process run by `BoxNarrow` may be very slow in some cases. The main idea of `PolyBox` is to work with two non-interval functions instead of $f_{[Y]}$. These non-interval

functions are obtained by a symbolic manipulation preprocessing that rewrites f into a new form g for which the two *extremal functions* enclosing $g_{[Y]}$ can be easily extracted. Then, during constraint propagation, `PolyBoxRevise` calls `BoxNarrow` on the two extremal functions of $g_{[Y]}$. This implies a faster contraction. In addition, when $g_{[Y]}$ is a low-degree polynomial, the computation of the new bounds of $[x]$ follows simple evaluations using the real roots of the extremal functions identified analytically.

2 Background

Intervals allow reliable computations on computers by managing floating-point bounds and outward rounding.

Definition 1 (Basic definitions, notations)

\mathbb{IR} denotes the set of **intervals** $[v] = [a, b] \subset \mathbb{R}$ where a , also denoted \underline{v} , and b , also denoted \overline{v} , are floating-point numbers. $\overline{v} - \underline{v}$ is the **size** of $[v]$.

An **interval vector**, or **box**, $[V] = ([v_1], \dots, [v_n])$ represents the Cartesian product $[v_1] \times \dots \times [v_n]$. Its size is the maximal size of its components $[v_i], i = 1, \dots, n$.

Interval arithmetic has been introduced to extend the real arithmetic to intervals [10]. For instance, we have straightforwardly $[v_1] + [v_2] = [\underline{v_1} + \underline{v_2}, \overline{v_1} + \overline{v_2}]$. This allows us to extend real valued functions to intervals. Such an **extension** must be defined so as to be conservative, i.e., $\forall V \in \mathbb{R}^k \ f(V) = [f](V)$ and $\forall [V] \in \mathbb{IR}^k \ [f]([V]) \supseteq \{f(V), V \in [V]\}$.

The **natural extension** $[f]_N$ of a real function f replaces arithmetic over the reals by interval arithmetic. Consider for instance the real function $f(x_1, x_2) = x_1^2 - 2x_1x_2 + x_2^2$. The natural extension $[f]_N$ from \mathbb{IR}^n to \mathbb{IR} is defined by $[f]_N([x_1], [x_2]) = [x_1]^2 - 2[x_1][x_2] + [x_2]^2$. Evaluated on the intervals $[x_1] = [x_2] = [0, 1]$, we obtain $[f]_N([x_1], [x_2]) = [-2, 2]$. Note that the natural extension of f depends upon its symbolic expression, and consequently is not unique. As a matter of fact, f may also be rewritten as $(x_1 - x_2)^2$ and yields the natural extension $([x_1] - [x_2])^2$. Note that $([x_1] - [x_2])^2 = [0, 1] = \{f(x_1, x_2), x_1 \in [x_1], x_2 \in [x_2]\} \subset [f]_N([0, 1], [0, 1])$. This illustrates the **dependency problem** which is a major concern in interval arithmetic. f has **multiple occurrences** of variables that are handled as different variables by interval arithmetic. In general, the dependency problem implies an overestimation of the interval image. It renders NP-hard the problem of finding the optimal interval image of a polynomial [8]. This raises the need to symbolic manipulations of expression before calculations so as to reduce overestimation.

The `PolyBox` algorithm presented in this paper aims at solving nonlinear systems of constraints or Numerical CSPs. An **NCSP** $P = (V, C, [V])$ contains a set of constraints C , a set V of n variables with domains $[V] \in \mathbb{IR}^n$. A solution $S \in [V]$ to P satisfies all the constraints in C . To approximate all the solutions of an NCSP with interval-based techniques, the solving process starts from an initial box representing the search space and builds a search tree, following a

Branch & Contract scheme. A *Branching* operation **bisects** the current box on one dimension (variable), generating two sub-boxes. At each node of the search tree, **contraction/filtering** algorithms improve the bounds of the current box with no loss of solutions. The process terminates with boxes of size smaller than a given positive ω .

The constraint programming community proposes constraint propagation algorithms that perform a propagation loop like AC3. Contracting optimally a box w.r.t. an individual constraint is referred to as **hull-consistency** problem. Similarly to the optimal interval image computation, due to the dependency problem, hull-consistency is not tractable. The main procedure of our algorithm is compared to two state-of-the-art *revise* algorithms that handle the constraints individually. **HC4-Revise** [2] is known to achieve the hull-consistency of constraints having *no* variable with multiple occurrences, provided that the function [1] is continuous. It traverses twice the tree representing the mathematical expression of the constraint for narrowing all the involved variable intervals. **BoxNarrow** [2,14] is stronger than **HC4-Revise** [4] and can enforce hull-consistency of a constraint when it contains *one* variable with multiple occurrences. In the general case, it enforces the *Box-consistency* property [2].

Definition 2 An NCSP $(X, C, [X])$ is **box-consistent** if every pair (c, x) , $c \in C$, $x \in X$ is box-consistent. Consider the pair (c, x) , where the constraint is described by $c : f(x, y_1, \dots, y_{k-1}) = 0$, $f : \mathbb{R}^k \rightarrow \mathbb{R}$, and the univariate interval function $f_{[Y]}(x) = f(x, [y_1], \dots, [y_{k-1}])$. The pair (c, x) is box-consistent (with respect to the natural extension $[f]_N$) on the domain $[x] = [\underline{x}, \bar{x}]$, if: $0 \in [f_{[Y]}]_N([\underline{x}, +])$ and $0 \in [f_{[Y]}]_N([-, \bar{x}])$, where $[\underline{x}, +]$ and $[-, \bar{x}]$ denote intervals of size one u.l.p. [2] at the bounds of $[x]$.

In practice, for every pair (f, x) , starting with an interval $[x]$, the **BoxNarrow** procedure returns a reduced interval $[x'] \subseteq [x]$ such that $[\underline{x}', +]$ (resp. $[-, \bar{x}']$) is the smallest (resp. largest) ϵ -solution [3] of the equation $f_{[Y]}(x) = 0$. Existing procedures use a *shaving* principle to narrow $[x]$: “Slices” $[\underline{x}, \underline{x} + \eta]$ (resp. $[\bar{x} - \eta, \bar{x}]$) are discarded from $[x]$ if $0 \notin [f_{[Y]}]_N([\underline{x}, \underline{x} + \eta])$ (resp. $0 \notin [f_{[Y]}]_N([\bar{x} - \eta, \bar{x}])$). This test sometimes uses a uni-variate interval Newton procedure.

Figure [1] illustrates that $f_{[Y]}$ is an interval function. It also shows the steps followed by **BoxNarrow**. The top (resp. the bottom) side of the figure details the “dichotomic” work performed by **LeftNarrow** (resp. **RightNarrow**) on slices/intervals of decreasing size, starting from $[x]$. For **LeftNarrow**, if the size of the current interval $[l]$ is less than or equal to 1 u.l.p. and $0 \in [f_{[Y]}]_N([l])$, then the procedure returns $[l]$. The last step 17 replaces the interval $[x]$ by the new interval $[L, \bar{r}]$. Observe that at the end of **RightNarrow** (step 16), $[r]$ does not contain any zero of the function but an ϵ -zero. The slicing performed by **BoxNarrow** on a variable x limits the overestimation effect on x , but not on the other variables y_i if they also occur several times.

¹ Along with *projection functions* used during the second top-down tree traversal...
² One Unit in the Last Place is the gap between two successive floating-point numbers.
³ $x \in \mathbb{R}^n$ is an ϵ -solution of $f(x) = 0$, if $[-\epsilon, \epsilon] \cap f(x) \neq \emptyset$.

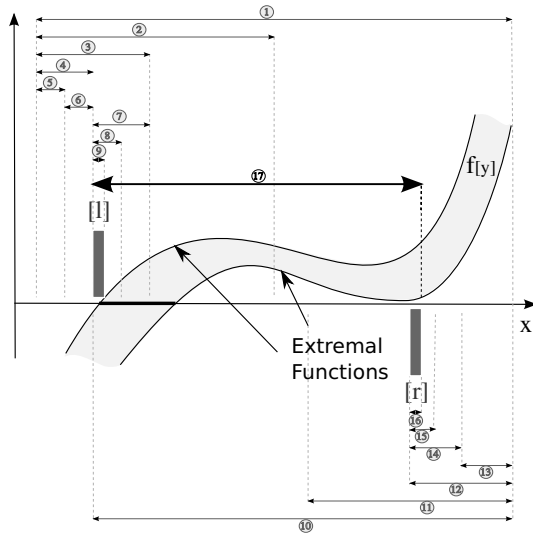


Fig. 1. The BoxNarrow procedure. The algorithm returns the interval computed in the step 17. (The additional contraction performed by Newton is not considered.)

3 Description of the PolyBoxRevise Procedure

The aim of our new PolyBoxRevise procedure is to limit the overestimation due to multiple occurrences of variables y_i and to speed up the iterative narrowing process introduced above. Before solving the system, in a preprocessing phase, for every pair (c, x) given by $c : f(x, y_1, \dots, y_{k-1}) = 0$, we use symbolic manipulation to rewrite f into a new form $g(x, y_1, \dots, y_{k-1})$. This preprocessing allows the PolyBoxRevise procedure to rapidly extract, during constraint propagation, non-interval *extremal functions* that enclose $g[y]$, before contracting $[x]$.

Symbolic Manipulation and Extremal Functions

For any pair (c, x) and any box $([x], [y_1], \dots, [y_{k-1}])$, the aim would be to extract the *optimal* extremal real (i.e., non-interval) functions $\underline{h}(x)$ and $\overline{h}(x)$ defined by $\underline{h}(x) = \min_{y_i \in [y_i]} f(x, y_1, \dots, y_{k-1})$ and $\overline{h}(x) = \max_{y_i \in [y_i]} f(x, y_1, \dots, y_{k-1})$, $\forall x \in [x]$. Then a necessary and sufficient condition for $x \in [x]$ to satisfy c is $0 \in [\underline{h}(x), \overline{h}(x)]$. Unfortunately, it is generally not tractable to determine \underline{h} and \overline{h} due to the overestimation implied by the dependency problem (see Section 2 and 8), and we have to be satisfied with functions \underline{g} and \overline{g} such that for any $x \in [x]$ we have $\underline{g}(x) \leq \underline{h}(x) \leq f(x, y_1, \dots, y_{k-1}) \leq \overline{h}(x) \leq \overline{g}(x)$, $\forall x \in [x], \forall y_i \in [y_i], i \in \{1, \dots, k-1\}$. Now the test $0 \in [\underline{g}(x), \overline{g}(x)]$ is only a necessary condition. We will refer to \underline{g} and \overline{g} as *minimal* and *maximal* extremal functions.

The aim is to *automatically* and rapidly identify extremal functions. This can clearly not be done for any function f and we have restricted our attention to the class of functions that can be described by

$$f(x, y_1, \dots, y_{k-1}) = \sum_{i=0}^d f_i(y_1, \dots, y_{k-1}) \cdot h_i(x)$$

where d is a positive integer and h_i has a finite number of zeros in $[x]$ that can be computed exactly. In addition, the sign of $h_i(x)$ is known for any $x \in [x]$. We have in mind elementary functions such as x^i , $\log(x)$ or e^x .

We have used the symbolic manipulation tool **Mathematica** [15] for automatically identifying functions f_i and h_i , and to rewrite them in the most appropriate manner. The procedure **FullSimplify** of **Mathematica** computes automatically several possible forms for every f_i (heuristically) and selects the form minimizing a given criterion. The criterion we have specified is the number of occurrences of each variable y_i . During the solving, like **BoxNarrow**, the **PolyBoxRevise** procedure first replaces, in the new analytic form g , the variables y_i by their domains. We thus obtain $g_{[Y]}(x) = \sum_{i=0}^d [f_i]_N([Y])h_i(x)$, $[Y] = ([y_1], \dots, [y_{k-1}])$. Given the box $[Y]$, the coefficients $[f_i]_N([Y])$, denoted $[c_i]$, are now numerical intervals. Due to the assumptions on h_i , the following two functions $\underline{g}_{[Y]}$ and $\overline{g}_{[Y]}$ are respectively minimal and maximal extremal functions, computed at the bounds of the interval coefficients $[c_i]$:

$$\underline{g}_{[Y]}(x) = \sum_{i=1}^d c_i^-(x)h_i(x) \quad \text{and} \quad \overline{g}_{[Y]}(x) = \sum_{i=1}^d c_i^+(x)h_i(x)$$

with $\begin{cases} c_i^-(x) = \underline{c}_i, & c_i^+(x) = \overline{c}_i, & \text{if } h_i(x) \geq 0 \\ c_i^-(x) = \overline{c}_i, & c_i^+(x) = \underline{c}_i, & \text{if } h_i(x) \leq 0 \end{cases}$

Example. Consider the function: $f(x, y_1, y_2) = (y_1 + y_2)x^2 + (2y_1y_2)x + \sin(y_2)$. For the domains $[y_1] = [0.5, 1]$, $[y_2] = [1, 2]$, we have $g_{[Y]}(x) = [1.5, 3]x^2 + [1, 4]x + [0.84147, 1]$ and then

$$\underline{g}_{[0.5,1],[1,2]}(x) = \begin{cases} 1.5x^2 + x + 0.84147, & \text{if } x \geq 0 \\ 1.5x^2 + 4x + 0.84147, & \text{if } x \leq 0 \end{cases}$$

$$\overline{g}_{[0.5,1],[1,2]}(x) = \begin{cases} 3x^2 + 4x + 1, & \text{if } x \geq 0 \\ 3x^2 + x + 1, & \text{if } x \leq 0 \end{cases}$$

Remark. $\underline{g}_{[Y]}$ and $\overline{g}_{[Y]}$ are optimal extremal functions of $g_{[Y]}$. However, although f and g are the same, the interval functions $f_{[Y]}$ and $g_{[Y]}$ are different because the replacement of the variables y_i by $[y_i]$, occurring several times in f , produce different overestimations. Hence, $\underline{g}_{[Y]}$ and $\overline{g}_{[Y]}$ constitute only approximate non-interval functions enclosing f . Also, Box-consistencies of $f_{[Y]}$ and $g_{[Y]}$ are not comparable. That is why our contractor starts by calling systematically the cheap **HC4-Revise** procedure on the initial form f before performing the process described below.

For a given $[Y]$, once the extremal functions have been determined, we proceed with the contraction part of **PolyBoxRevise** (during constraint propagation). Starting with an initial interval $[x]$, let us detail how the new and improved left bound \underline{l} ($[l]$ is 1 u.l.p. large) of $[x]$ is determined. (A symmetric process is performed for the right bound.) **PolyBoxRevise** first determines with which extremal function to work with. Three cases occur:

1. If $g_{[Y]}(\underline{x}) \leq 0$ and $0 \leq \overline{g_{[Y]}(\underline{x})}$: $l = \underline{x}$ (no contraction)
2. If $\overline{g_{[Y]}(\underline{x})} > 0$: $g_{[Y]}$ is selected
3. If $\overline{g_{[Y]}(\underline{x})} < 0$: $\overline{g_{[Y]}}$ is selected (situation depicted in the left side of Fig. [11](#))

The smallest root $[l]$ of $\overline{g_{[Y]}(x)} = 0$ in $[x]$ can now be computed using the standard **BoxNarrow** (i.e., **LeftNarrow**) procedure applied to the extremal function selected. The advantage is a faster convergence since **BoxNarrow** is run with a non-interval function.

We have implemented the polynomial case, where $h_i(x) = x^i, i = 0, \dots, d$. In particular, when the degree d is smaller than 4, instead of using **BoxNarrow** to determine the real roots of $\overline{g_{[Y]}(x)} = 0$, we have used explicit analytical expressions of the roots. For $d = 3$, we have used the Cardano's expressions⁴ of the real roots. We have adapted these symbolic methods to manage rounding errors due to floating point calculation by first replacing all the coefficients by a degenerate interval (of null size).

Finally, we have implemented a new procedure **PolyBoxRevise** based on the **Box** algorithm variant called **BC4** [2](#). If $f(x, y_1, \dots, y_{k-1})$ has a single occurrence of x , **PolyBoxRevise** calls **HC4-Revise** (like **BC4** does). Otherwise, it uses the rewritten form g (with appropriate symbolic expressions for the $f_i(y_1, \dots, y_{k-1})$) of f produced *automatically* by the **FullSimplify** procedure of **Mathematica** [15](#) in the preprocessing. Four cases occur:

1. f is not polynomial w.r.t. x : the procedure calls **BoxNarrow** (or **HC4-Revise** in a hybrid version because it is less time consuming).
2. $g_{[Y]}(x)$ contains only one occurrence of x : **HC4-Revise** is applied to $g_{[Y]}(x)$.
3. $g_{[Y]}(x)$ has multiple occurrences of x and $d < 4$:
analytic determination of the smallest root of $g_{[Y]}(x)$ in $[x]$.
4. $g_{[Y]}(x)$ has multiple occurrences of x and $d \geq 4$:
numerical determination of the smallest root of $g_{[Y]}(x)$ in $[x]$, using **BoxNarrow**.

Remarks. The second case above can be illustrated by an equation of the system **Caprasse** (tested below): $-2x + 2txy - z + y^2z = 0$ that our symbolic tool rewrites into: $(-2 + 2ty)x + (-1 + y^2)z = 0$ (for the contraction of $[x]$ or $[z]$). Observe that the new form makes disappear the multiple occurrences of x and z . The decrease in occurrences of x and z illustrates a successful transformation leading to a gain in CPU time. An equation of the instance **6body** shows a counterproductive transformation of $5(b_1 - d_1) + 3(b_2 - d_2)(b_1 + d_1 - 2f_1) = 0$ into $b_1(5 + 3(b_2 - d_2)) + (-5 + 3b_2 - 3d_2)d_1 + 6(-b_2 + d_2)f_1 = 0$. Indeed, for obtaining an expanded form on b_1 or d_1 , the transformation increases the overestimation because of the additional occurrences of variables a_2, b_2 and d_2 .

Comparison with the Box Algorithm of Numerica

Van Hentenryck, Michel et Deville have also used extremal functions (without using this vocabulary) in their interval-based solver **Numerica** [14](#). The principle

⁴ G. Cardano. *Ars magna, sive de regulis algebraicis liber unus*, Nuremberg, 1545.

is introduced in one page in a technical article [13]. *Numerica* manages different forms of the handled system, and a separate constraint propagation is run on the system in an entirely expanded form for using extremal functions.

PolyBox follows on the contrary a scheme close to BC4. It manages a unique system with revise procedures adapted to every pair (f, x) , which causes an overestimation smaller than the entirely expanded form used by *Numerica*. In addition, like BC4, *PolyBox* also uses HC4-Revise when x occurs only once in f . Finally, the analytic solving of low degree polynomials is added.

4 Experiments

We have compared our *PolyBox* algorithm to BC4 and HC4. The symbolic manipulation of all pairs (f, x) is achieved in a fraction of a second in a pre-processing by *Mathematica* [15]. All the contractors have been implemented in the free *Ibex* interval-based C++ library [3]. To find all the solutions to the tested NCSPs, the solving strategy bisects the variables in a round-robin way. Between two branching points in the search, constraint propagation (i.e., *PolyBox*, HC4 or BC4) is performed before an interval Newton.

Table 1. Results. The entries in the last four columns are the CPU time in second (first row) and the number of nodes in the search tree (second row).

Name	#var	#sol	HC4	BC4	PolyBox--	PolyBox
Caprasse	4	18	5.53	37.2	2.34	2.16
			9539	6509	2939	2939
Yamamura1	8	7	34.3	13.4	5.79	2.72
			42383	4041	2231	2231
Extended Wood	4	3	0.76	1.94	1.34	1.12
			4555	1947	3479	3479
Broyden Banded	20	1	> 3600	0.62	0.16	0.09
			?	1	1	1
Extended Freudenstein	20	1	> 3600	0.19	0.22	0.11
			?	121	121	121
6body	6	5	0.58	2.93	0.73	0.73
			4899	4797	4887	4887
Rose	3	18	> 3600	> 3600	4.00	4.10
			?	?	12521	12521
Discrete Boundary	39	1	179	29.5	41.8	16.1
			185,617	3279	3281	3281
Katsura	12	7	102	404	103	104
			14007	11371	13719	13719
Eco9	8	16	66	191	71	71
			132,873	125,675	131,911	131,911
Broyden Tridiagonal	20	2	470	495	403	350
			269,773	163,787	164,445	164,445
Geneig	6	10	3657	> 7200	3508	3363
			79,472,328	?	4,907,705	4,907,705

Among the 44 polynomial systems with isolated solutions found in COPRIN's Web page⁵, we have selected the 12 instances that are solved by at least one of the 3 strategies in a time comprised between 1 second and 1 hour (on a Pentium 3 GHz) and have equations with multiple occurrences of the variables.

Table 1 reports interesting speedups brought by PolyBox on these instances. The column PolyBox-- in the table corresponds to a variant of PolyBox in which the low degree polynomials are not handled analytically but by BoxNarrow. The additional gain brought by the analytic process is significant in only two NCSPs.

Our first results are promising, so that it should be worthwhile hybridizing PolyBox with other algorithms, especially those achieving the Box-consistency or a weaker form of it [15]. An idea would be to keep the rewritten forms only if they are of degrees 2 and 3, and add them as global and redundant constraints in the system for improving the constraint propagation.

References

1. Araya, I., Trombettoni, G., Neveu, B.: Making Adaptive an Interval Constraint Propagation Algorithm Exploiting Monotonicity. In: Proc. CP. LNCS (2010)
2. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.-F.: Revising Hull and Box Consistency. In: Proc. ICLP, Conf. on Logic Programming, pp. 230–244 (1999)
3. Chabert, G.: Ibex – An Interval Based EXplorer (2010), <http://www.ibex-lib.org>
4. Collavizza, H., Delobel, F., Rueher, M.: Extending Consistent Domains of Numeric CSP. In: Proc. IJCAI, pp. 406–413 (1999)
5. Goldsztejn, A., Goualard, F.: Box Consistency through Adaptive Shaving. In: Proc. ACM SAC, pp. 2049–2054 (2010)
6. Jaulin, L.: Interval Constraint Propagation with Application to Bounded-error Estimation. *Automatica* 36, 1547–1552 (2000)
7. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht (1996)
8. Kreinovich, V., Lakeyev, A.V., Rohn, J., Kahl, P.T.: Computational Complexity and Feasibility of Data Processing and Interval Computations. Kluwer, Dordrecht (1997)
9. Merlet, J.-P.: Interval Analysis and Robotics. In: Symp. of Robotics Research (2007)
10. Moore, R.E.: Interval Analysis. Prentice-Hall, Englewood Cliffs (1966)
11. Rueher, M., Goldsztejn, A., Lebbah, Y., Michel, C.: Capabilities of Constraint Programming in Rigorous Global Optimization. In: NOLTA (2008)
12. Tucker, W.: A Rigorous ODE Solver and Smale's 14th Problem. *Found. Comput. Math.* 2, 53–117 (2002)
13. Van Hentenryck, P., McAllester, D., Kapur, D.: Solving Polynomial Systems Using a Branch and Prune Approach. *SIAM J. on Num. Analysis* 34(2) (1997)
14. Van Hentenryck, P., Michel, L., Deville, Y.: *Numerica: A Modeling Language for Global Optimization*. MIT Press, Cambridge (1997)
15. Wolfram, S.: *Mathematica*, 4th edn. Cambridge University Press, Cambridge (1999)

⁵ See www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html. These benchmarks have been proposed by the interval community and some of them correspond to real problems.

Exponential Propagation for Set Variables

Justin Yip and Pascal Van Hentenryck

Brown University, Box 1910, Providence, RI 02912, USA

Abstract. Research on constraint propagation has primarily focused on designing polynomial-time propagators sometimes at the cost of a weaker filtering. Interestingly, the evolution of constraint programming over sets have been diametrically different. The domain representations are becoming increasingly expensive computationally and theoretical results appear to question the wisdom of these research directions. This paper explores this apparent contradiction by pursuing even more complexity in the domain representation and the filtering algorithms. It shows that the product of the length-lex and subset-bound domains improves filtering and produces orders of magnitude improvements over existing approaches on standard benchmarks. Moreover, the paper proposes exponential-time algorithms for NP-hard intersection constraints and demonstrates that they bring significant performance improvements and speeds up constraint propagation considerably.

1 Introduction

Constraint programming (CP) is often seen as a computational methodology for tackling constraint satisfaction problems (CSPs) characterized by the slogan

$$\text{Constraint Programming} = \text{Filtering} + \text{Search}.$$

Since most CSPs are NP-complete, CP uses filtering algorithms and constraint propagation to reduce the variable domains and hence the search tree to explore. The hope is that the reduction in the search space is sufficient to solve problems of interest in reasonable time. In general, researchers have focused on designing polynomial-time algorithms for filtering, leaving the potentially exponential behavior in the search component. There are exceptions of course, and we will review some of them later, but researchers overwhelmingly focus on polynomial-time filtering algorithms, sometimes at the expenses of enforcing arc or bound consistency. This paper takes the other road and argues that exponential filtering algorithms and constraint propagation may be highly beneficial in practice. It is motivated by the fact that reasonable exponential behavior in the filtering algorithm may produce significant reduction of search space and can therefore be cost-effective. Moreover, such a reasonable exponential behavior has beneficial effects on constraint propagation allowing further reduction of the search space, an observation made by Bessiere and Régin in [1] where they solve CSPs on the fly to achieve arc consistency on a global constraint. Finally, since the overall approach is exponential in the worst case, it may be preferable to shift

some of the exponential behavior from the largely agnostic search to the filtering component where we can exploit the semantics of the constraints and locality.

This paper evaluates this idea in the context of CSPs over set variables. Set variables are natural objects for modeling classes of combinatorial problems but they raise fundamental computational issues. Since their domains often represent an exponential number of sets, maintaining an explicit enumeration of the domain values is computationally expensive and hence much effort has been devoted to approximating a set domain or using a compact and precise representation. The subset-bound domain was introduced in the early 90s and is now available in many CP-solvers (e.g., [2,3]). It maintains two sets r, p and defines its domain as $\{s \mid r \subseteq s \subseteq p\}$. The subset-bound domain supports polynomial-time filtering algorithms for a variety of constraints but the pruning it offers is generally weak¹. As a result, much research has been devoted to design richer representations of set domains. Sometimes the subset-bound domain is enhanced with a cardinality component to restrict the cardinality of the set variable [4,5], which often results in much stronger propagation. However, Bessiere and al. in [6] have shown that some natural global constraints become intractable when moving to this domain. Some exact representations of sets use ordered binary decision diagrams and their propagation algorithms can obviously take exponential time [7]. The length-lex representation takes a dual perspective by primarily capturing the cardinality and lexicographical information [8]. A length-lex domain is defined by two bounds l, u as $\{s \mid l \preceq s \preceq u\}$ where \preceq is the (total) length-lex ordering. Here the filtering algorithms for many elementary constraints are typically polynomial-time but the constraint-propagation algorithm may take exponential time to converge to a fixpoint [9]². There is thus an abundance of negative theoretical results on richer set representations. Yet, on a wide variety of standard benchmarks, the richer representations bring orders of magnitude improvements compared to the subset-bound domain or traditional encodings in terms of finite-domain variables [7,14].

The goal of this paper is thus to explore whether it is beneficial to boost constraint propagation over set variables even further. It explores two ideas: the product of the length-lex and subset-bound domains which has never been evaluated before and exponential propagators for intersection constraints. Its main contributions are as follows:

1. It shows that the propagation of simple unary constraints in the length-lex domain may take exponential time to converge, yet the length-lex domain brings orders of magnitude improvements over other representations.
2. It demonstrates that the product of the subset-bound and length-lex domains brings additional pruning on intersection constraints and significant improvements in efficiency.

¹ Note that, even with this compact representation, many intractable problems can be encoded by a unary constraint over a single set variable.

² The fact that constraint-propagation algorithms may take exponential time to converge is not specific to set variables. It appears for instance in numerical continuous CSPs (e.g., [12]) and the propagation of cumulative constraints [13].

3. It proves the $W[1]$ -Hardness and NP-completeness of unary intersection constraint for the subset-bound with cardinality and length-lex domains.
4. It proposes exponential but complete filterings for these intersection constraints and shows that they bring again an order of magnitude improvement in efficiency compared to existing approaches.
5. It shows that these exponential filtering algorithms speeds up the convergence of the constraint propagation algorithm considerably.

The rest of the paper is organized as follows. Section 2 reviews common set representations and formally specifies the consistency notion to establish the theoretical results of the paper. Section 3 discusses intractability issues for set variables. Section 4 contrasts the theoretical results with an experimental evaluation of the various domains. Section 5 proposes an exponential-time propagator for a $W[1]$ -hard unary intersection and Section 6 evaluates the effectiveness and efficiency of the proposed constraint. Section 7 discusses some of the related work. Section 8 concludes the paper.

2 Domain Representations and Consistency Notions

This section reviews the domain representation for set variables and the consistency notions for set constraints.

Notations. For simplicity, we assume that sets take their values in a universe $U(n)$ of integers $\{1, \dots, n\}$ equipped with traditional set operations. n, m are integers denoting the size of a universe, the number of variables, or both. X , possibly subscripted, is a set variable. Elements of $U(n)$ are denoted by the letter e . Sets are denoted by l, u, s, r, p . A subset s of $U(n)$ of cardinality c is called c -set and is denoted as $\{s_1, s_2, \dots, s_c\}$ where $s_1 < s_2 < \dots < s_c$. The length-lex ordering \preceq , proposed in [8], totally orders sets first by cardinality and then lexicographically.

Definition 1 (Length-Lex Ordering). *The length-lex ordering is defined by*

$$s \preceq t \text{ iff } s = \emptyset \vee |s| < |t| \vee |s| = |t| \wedge (s_1 < t_1 \vee s_1 = t_1 \wedge s \setminus \{s_1\} \preceq t \setminus \{t_1\})$$

Its strict version is defined by $s \prec t$ iff $s \preceq t \wedge s \neq t$.

Example 1. Given $U(4) = \{1, \dots, 4\}$, we have $\emptyset \prec \{1\} \prec \{2\} \prec \{3\} \prec \{4\} \prec \{1, 2\} \prec \{1, 3\} \prec \{1, 4\} \prec \{2, 3\} \prec \{2, 4\} \prec \{3, 4\} \prec \{1, 2, 3\} \prec \{1, 2, 4\} \prec \{1, 3, 4\} \prec \{2, 3, 4\} \prec \{1, 2, 3, 4\}$.

Definition 2 (ll-domain). *A length-lex domain (ll-domain) is a pair of sets $ll\langle l, u, n \rangle$. It contains all sets (inclusively) in the universe $U(n)$ between l and u in the length-lex ordering:*

$$ll\langle l, u, n \rangle \equiv \{s \subseteq U(n) \mid l \preceq s \preceq u\}.$$

Example 2. The length-lex domain $ll\langle\{1, 3, 8\}, \{1, 5, 8\}, 8\rangle$ denotes the set $\{\{1, 3, 8\}, \{1, 4, 5\}, \{1, 4, 6\}, \{1, 4, 7\}, \{1, 4, 8\}, \{1, 5, 6\}, \{1, 5, 7\}, \{1, 5, 8\}\}$.

Definition 3 (ll-bound consistency). A set constraint $C(X_1, \dots, X_m)$ (X_i are set variables using the ll-domain) is said to be ll-bound consistent if and only if $\forall 1 \leq i \leq m$,

$$\begin{aligned} & \exists x_1 \in d(X_1), \dots, x_m \in d(X_m) \text{ s.t. } C(x_1, \dots, x_{i-1}, l_{X_i}, x_{i+1}, \dots, x_m) \\ & \wedge \exists x_1 \in d(X_1), \dots, x_m \in d(X_m) \text{ s.t. } C(x_1, \dots, x_{i-1}, u_{X_i}, x_{i+1}, \dots, x_m) \end{aligned}$$

where $d(X_i) = ll\langle l_{X_i}, u_{X_i}, n_{X_i} \rangle$ denotes the domain of X_i .

Enforcing ll-bound consistency assures that the bounds of each variable appears in a solution to the constraint, which corresponds to the traditional notion of bound consistency on finite-domain variables. It is a rather strong property for set domains and is well-defined because the length-lex ordering is total.

Definition 4 (sbc-domain). A subset-bound + cardinality domain (sbc-domain) $sbc\langle r, p, \check{c}, \hat{c} \rangle$ consists of a required set r and a possible set p , a minimum and maximum cardinalities \check{c} and \hat{c} , and represents the set of sets

$$sbc\langle r, p, \check{c}, \hat{c} \rangle \equiv \{s \mid r \subseteq s \subseteq p \wedge \check{c} \leq |s| \leq \hat{c}\}$$

Example 3. The sbc-domain $sbc\langle\{1\}, \{1, 3, 4, 5, 7, 8\}, 3, 3\rangle$ denotes the set $\{\{1, 3, 4\}, \{1, 3, 5\}, \{1, 3, 7\}, \{1, 3, 8\}, \{1, 4, 5\}, \{1, 4, 7\}, \{1, 4, 8\}, \{1, 5, 7\}, \{1, 5, 8\}, \{1, 7, 8\}\}$.

Definition 5 (sbc-bound consistency). A set constraint $C(X_1, \dots, X_m)$ (X_i are set variables using the sbc-domain) is said to be sbc-bound consistent if and only if $\forall 1 \leq i \leq m$,

$$\begin{aligned} & \exists x_1 \in d(X_1), \dots, x_m \in d(X_m) \text{ s.t. } C(x_1, \dots, x_m) \\ & \wedge r_{X_i} = \bigcap_{\forall 1 \leq j \leq m, x_j \in d(X_j): C(x_1, \dots, x_m)} x_i \wedge p_{X_i} = \bigcup_{\forall 1 \leq j \leq m, x_j \in d(X_j): C(x_1, \dots, x_m)} x_i \\ & \wedge \exists x_1 \in d(X_1), \dots, x_m \in d(X_m) \text{ s.t. } |x_i| = c_{\check{X}_i} \wedge C(x_1, \dots, x_m) \\ & \wedge \exists x_1 \in d(X_1), \dots, x_m \in d(X_m) \text{ s.t. } |x_i| = c_{\hat{X}_i} \wedge C(x_1, \dots, x_m) \end{aligned}$$

where $d(X_i) = sbc\langle r_{X_i}, p_{X_i}, c_{\check{X}_i}, c_{\hat{X}_i} \rangle$ denotes the domain of X_i .

Note that this definition requires the constraint to have a solution consistent with the domains of the variables (first condition) and to have solutions consistent with each of the cardinality bounds of the variables (fourth and fifth conditions). We now define the product of the length-lex and subset-bound domain. (Similar hybrid domain representations were proposed in [10,11].)

Definition 6 (ls-domain). A length-lex + subset-bound domain (ls-domain) is the intersection of the two aforementioned domains. A ls-domain $ls\langle l, u, n, r, p \rangle$ consists of two bounds l, u for the length-lex ordering, a universe size n , a required set r , and a possible set p . It represents the set of sets

$$ls\langle l, u, n, r, p \rangle \equiv ll\langle l, u, n \rangle \cap sbc\langle r, p, |l|, |u| \rangle$$

Example 4. The ls-domain $ls\langle\{1, 3, 8\}, \{1, 5, 8\}, 8, \{1\}, \{1, 3, 4, 5, 7, 8\}\rangle$ (which is the intersection of domains in Example 2 and 3) denotes the set $\{\{1, 3, 8\}, \{1, 4, 5\}, \{1, 4, 7\}, \{1, 4, 8\}, \{1, 5, 7\}, \{1, 5, 8\}\}$.

Definition 7 (ls-bound consistency). A set constraint $\mathcal{C}(X_1, \dots, X_m)$ (X_i are set variables using the ls-domain) is said to be ls-bound consistent if and only if $\forall 1 \leq i \leq m,$

$$\begin{aligned}
 & l_{X_i} \in d(X_i) \wedge \exists x_1 \in d(X_1), \dots, x_m \in d(X_m) : C(x_1, \dots, x_{i-1}, l_{X_i}, x_{i+1}, \dots, x_m) \\
 & \wedge u_{X_i} \in d(X_i) \wedge \exists x_1 \in d(X_1), \dots, x_m \in d(X_m) : C(x_1, \dots, x_{i-1}, u_{X_i}, x_{i+1}, \dots, x_m) \\
 & \wedge r_{X_i} = \bigcap_{\forall 1 \leq j \leq m, x_j \in d(X_j) : C(x_1, \dots, x_m)} x_i \wedge p_{X_i} = \bigcup_{\forall 1 \leq j \leq m, x_j \in d(X_j) : C(x_1, \dots, x_m)} x_i
 \end{aligned}$$

where $d(X_i) = ls\langle l_{X_i}, u_{X_i}, n_{X_i}, r_{X_i}, p_{X_i}\rangle$

We now show that the ls-domain is strictly stronger than the conjunction of the ll-domain and sbc-domain.

Lemma 1. Enforcing bound consistency on a ls-domain is strictly stronger than enforcing bound consistency separately on the decomposition of the ll-domain and sbc-domain.

Proof. Clearly enforcing bound consistency on the ls-domain is at least as strong. Consider a unary constraint $|X \cap \{4, 5, 7\}| \leq 1$ and the ls-domain in Example 4. It is bound-consistent for the decomposition, since the lower and upper bounds satisfy the constraint and the required and possible sets are bound-consistent. However, for the ls-domain, only three domain values, i.e., $\{1, 3, 8\}, \{1, 4, 8\}, \{1, 5, 8\}$, satisfy the constraint. Element 8 belongs to all solutions and thus to the required set. Enforcing bound consistency for the ls-domain yields $ls\langle\{1, 3, 8\}, \{1, 5, 8\}, 8, \{1, 8\}, \{1, 3, 4, 5, 7, 8\}\rangle$. \square

3 Theoretical Results on Intersection Constraints

We now present a number of theoretical results, which shed light on the behavior and complexity of filtering algorithms and constraint propagation on set domains. In the following, we use $bc_\theta\langle\mathcal{C}\rangle$ to denote a bound-consistency propagator and $hs_\theta\langle\mathcal{C}\rangle$ to denote a feasibility routine for constraint \mathcal{C} on a θ -domain.

The first result we mentioned is well-known but quite interesting and concerns the sbc-domain.

Theorem 1. $hs_{sbc}\langle |X_i \cap X_j| \leq 1, \forall i < j \rangle$ is NP-hard. [6]

We consider a special case of this constraint in which all but one variables are bounded. We show that, even in this simple unary case, enforcing bound consistency on both the sbc-domain and the ll-domain is fixed-parameter intractable.

Definition 8 (atmost1). $atmost1(\{s_1, \dots, s_m\}, X) \equiv |X \cap s_i| \leq 1, \forall 1 \leq i \leq m$

Theorem 2. $hs_{sbc}\langle atleast1(\{s_1, \dots, s_m\}, X) \rangle$ is NP-hard.

Proof. Reduction from k-Independent Set. Instance: Graph $G = (V, E)$ and a positive integer $k \leq |V|$. Question: Does G contains an independent set of size k , i.e. a k -subset V' of V such that no two vertices in V' join by an edge in E .

We construct an instance of CSP with one sbc-variable X and one constraint $atleast1(\{s_1, \dots, s_m\}, X)$. Intuitively, X corresponds to a independent set and each set s_i corresponds to the neighborhood of vertex i and itself. Hence, X can take at most 1 element from each set corresponds to the restriction that no two vertices in the independent set join by an edge.

Formally, for every $i \in V$, $s_i = \{i\} \cup adj(i)$ (where $adj(i)$ denotes the neighborhood of vertex i), and $X \in sbc\langle \emptyset, V, k, k \rangle$. The CSP has a solution if and only if G has a independent set of size k . \Rightarrow Given a k independent set V' , we can construct a solution by setting $X = V'$ since every element in X actually corresponds to a vertex. When X takes an element i , since the size of intersection is at most 1, it cannot take any other element from set s_i (i.e. $adj(i)$), the definition of independent set guarantees this. \Leftarrow Given a consistent assignment of X , it is a independent set since any edge corresponds to taking two element from the same set which violates the $atleast1$ constraint. \square

Corollary 1. $hs_{sbc}\langle atleast1(\{s_1, \dots, s_m\}, X) \rangle$ is W[1]-hard.

Proof. k-Independent Set is a W[1]-Complete problem. [15] \square

Corollary 2. $hs_{ll}\langle atleast1(\{s_1, \dots, s_m\}, X) \rangle$ is W[1]-hard.

Proof. For any sbc-domain that contains only all k -sets of some universe, there exists an equivalent ll-domain $sbc\langle \emptyset, V, k, k \rangle \equiv ll\langle \Delta_k, \nabla_k, |V| \rangle$ with $\Delta_k = \min_{\preceq} \{s \mid s \subseteq V \wedge |s| = k\}$ and $\nabla_k = \min_{\preceq} \{s \mid s \subseteq V \wedge |s| = k\}$. \square

This result has an interesting corollary. Consider the propagation of a set of unary constraints of the form $|X \cap s_i| \leq 1$ ($1 \leq i \leq n$). These constraints enjoy a polynomial-time bound-consistency algorithm in the length-lex domain. By definition of bound consistency, constraint propagation terminates in a failure or in a state where the bounds of the variable are solutions. Hence, by Corollary 2, constraint propagation cannot run in time $O(f(k)n^{O(1)})$ in the worst case.

Corollary 3. The propagation algorithm for a collection of $bc_{ll}\langle |X \cap s_i| \leq 1 \rangle$ over X cannot run in time $O(f(k)n^{O(1)})$ in the worst case unless $FPT = W[1]$.

Similar results hold for other intersection constraints.

Definition 9 (exact1). $exact1(\{s_1, \dots, s_m\}, X) \equiv |X \cap s_i| = 1, \forall 1 \leq i \leq m$

Theorem 3. $hs_{sbc}\langle exact1(\{s_1, \dots, s_m\}, X) \rangle$ is NP-hard.

Proof. Reduction from 1-in-3 SAT. Instance: Set of n variables and m clauses, where each clauses consists of exactly three literals and each literal is either

a variable or its negation. Question: Does there exist a truth assignment to variables such that each clause has exactly one true literal?

Given an instance of 1-in-3 SAT, we construct a CSP with a *exact1* constraint. A set variable X associated with a sbc-domain $sbc(\emptyset, \{1, -1, \dots, n, -n\}, n, n)$ corresponds to a truth assignment. $i \in X$ means variable i is true and vice versa. There are two types of sets. Set $s_i = \{i, -i\}$ ($1 \leq i \leq n$) ensures a variable can either be true or false. Set $t_j = \{p, -q, r\}$ corresponds to a clause $(x_p \vee \neg x_q \vee x_r)$ guarantees that exactly one of its literal is true. Hence, we post the constraint $exact1(\{s_1, \dots, s_n, t_1, \dots, t_m\}, X)$. Clearly, the input instance has feasible assignment if and only if the CSP has a solution. \square

Similarly, a NP-hardness proof for the feasibility routine of *atleast1* is obtained from Theorem 3 by changing the input instance to 3SAT.

As mentioned earlier, the potentially exponential behavior of constraint propagation was pointed out in [9] for knapsack constraints and similar results exist for continuous constraints and edge-finding algorithms for cumulative constraints. What is somewhat surprising here is the simplicity of the constraint involved, which are simple unary intersection constraints. *This abundance of negative theoretical results may lead researchers to conclude that the sbc-domain and, even more so, the ll-domain are unworthy of any consideration. Experimental results however clearly indicate otherwise as we now discuss.*

4 Experimental Behavior of Domain Representations

The experimental results in this section compare the ll-domain and ls-domain, as well as the hybrid BDD-SAT recently proposed in [16] which provides state-of-the-art results in this area. The models use efficient complete filtering algorithms for binary constraints on these different domains [16, 17, 18, 14, 19]. We evaluate them on two standard CSP benchmarks: the social golfer problem and the steiner triple system. These models run on a 2.4GHz Core 2 Duo laptop with 4GB of memory. The time limit is 1800 seconds and \times indicates a timeout. Note that these are the first experimental results on the product of the length-lex and subset-bound with cardinality domains.

Social Golfer Problem. The ll-domain uses the model and propagators given in [17, 14]. The sbc-domain uses the model in [20] and the *pairatmost1* propagator in [18]. The ls-domain uses both models, with a channeling constraint to link the corresponding variables in the two models. The models use the same static labeling technique: variables are labeled in a week-wise fashion and, within each week, the variable with the largest domain is selected first (ties are broken by smaller group index first), and the choice consists in inserting the smallest element first and to exclude it on backtracking. BDD-SAT-VSIDS encodes the set domains with binary decision diagrams (BDDs) which are then integrated with a SAT solver for clauses generation and no-good learning. The SAT formulation employs a VSIDS search [21] that features a variable selection heuristics and

Table 1. The Behavior of Set Domains on the Social Golfer Problem

g,s,w	ll-domain		ls-domain		BDD-SAT-VSIDS
	Time	Fails	Time	Fails	Time
48 Easy	1.52	181	1.38	137	6.14
4,3,5	0.05	69	0.04	64	0.46
5,3,6	3.93	2728	2.47	1991	0.8
5,3,7	17.06	7650	12.1	6274	6.32
5,4,5	0.35	218	0.19	147	0.98
5,5,4	0.17	89	0.14	77	0.05
5,5,5	0.25	87	0.22	75	0.07
5,5,6	0.21	60	0.17	47	0.15
5,5,7	0.01	1	0.01	1	3.28
6,5,5	51.52	17197	21.61	9948	15.24
7,6,2	0.04	14	0.03	14	1.44
10,3,9	1.15	82	0.77	5	16.66
10,3,10	1.52	82	1.08	12	110.8
Total	77.77	28458	40.2	18792	162.39

restarts. BDD-SAT-VSIDS runs on a 3.00GHz Core 2 Duo with 2 GB of memory. Table 1 gives a comprehensive comparison between these approaches on the social golfer problem. The ls-domain explores the smallest search tree and is the most robust overall. It is about twice as fast as the ll-domain and 4 times faster than BDD-SAT-VSIDS, which provides state-of-the-art results. The ll-model is also more than twice as fast as the BDD-SAT-VSIDS.

The Steiner Triple System. The steiner triple system with n elements is a special class of balanced incomplete block design by setting $v = n, b = n(n - 1)/6, r = (n - 1)/2, k = 3, \lambda = 1$ [22]. It can be modeled using a dual set of set variables in which a primal variable X_i ($1 \leq i \leq v$) corresponds to a point and a dual variable Y_j ($1 \leq j \leq b$) corresponds to a block. In steiner triple system, every pair of points have exactly one common element (i.e., $|X_i \cap X_j| = 1, \forall i \neq j$). Complete filtering algorithm for the ll-domain is given in [17]. For the sbc-domain, the problem can be expressed as a conjunction of *pairatmost1* [18] and *nonEmptyIntersection* [19]. The ls-domain uses both model with channeling constraints to link them. All models apply a static smallest-index first labeling technique in the primal variable. For the BDD-SAT formulation, two search strategies are given: a static labeling and the VSIDS search discussed above.

Table 2 compares different approaches on Steiner Triple System. In the BDD-SAT formulation, \times indicates a timeout of 900 seconds, and unreported instances are left as blanks. The ls-domain is the fastest in average and on most instances. It may bring significant benefits compared to the ll-domain and, especially over BDD-SAT. In particular, the BDD-SAT formulation appears much less robust. Once again, the experimental results indicate that the rich ls-domain provide state-of-the-art performance, despite its potentially high complexity.

Table 2. The Behavior of Set Domains on Steiner Triple System

n	ll-domain		ls-domain		BDD-SAT-static	BDD-SAT-VSIDS
	Time	Fails	Time	Fails	Time	Time
7	0.01	0	0.01	0	0.01	0.03
9	0.01	1	0.01	1	0.02	0.02
13	0.01	1	0.02	51	0.06	0.02
15	0.03	1	0.04	73	0.07	0.32
19	0.12	6	0.17	140	0.37	0.07
21	0.29	30	0.35	192	0.82	39.19
25	4.84	943	5.63	912	7.1	×
27	14.01	650	12.61	1826	12.88	229.59
31	2.86	0	2.03	0	5.38	×
33	35.94	627	24.32	1714	443.07	19.3
37	×	×	1345.34	16178		
39	1001.74	12040	505.57	5248		

Algorithm 1. $bc_{ls}\langle atleast1(\{s_1, \dots, s_m\}) \rangle(X_{ls} = ls\langle l, u, n, r, p \rangle)$

- 1: $\mathcal{S} \leftarrow \{s \in X_{ls} \mid \bigwedge_{1 \leq i \leq m} |s \cap s_i| \leq 1\}$
 - 2: $l', u' \leftarrow \min_{\prec} \mathcal{S}, \max_{\prec} \mathcal{S}$
 - 3: $r', p' \leftarrow \bigcap_{s \in \mathcal{S}} s, \bigcup_{s \in \mathcal{S}} s$
 - 4: **return** $ls\langle l', u', n, r', p' \rangle$
-

5 Exponential Filtering for Intersection Constraints

The previous sections reported intriguing theoretical and experimental results. The theory indicated that constraint propagation of even simple constraints may take exponential time in the worst case for the length-lex domain, while the experimental results clearly showed that the product of length-lex and subset-bound domains led to the best and most robust performance on some standard benchmarks. In this section, we reconsider the intractable unary intersection constraint. Instead of decomposing them into simpler unary constraints, we propose simple exponential algorithms for enforcing bound consistency on the ll- and ls-domains. Our motivation is twofold:

1. An exponential filtering algorithm enables us to move the potentially exponential behavior from the rather agnostic constraint propagation algorithm into the constraint itself where the constraint semantics can be exploited.
2. The stronger filtering further increases the pruning of the search and may lead to additional domain reduction through constraint propagation of other constraints, an observation already pointed out in [1].

Algorithm 1 implements $bc_{ls}\langle atleast1(\{s_1, \dots, s_m\}, X) \rangle$ and is self-explanatory. The set \mathcal{S} maintains a logical enumeration of all possible solutions. All four bounds of the ls-domain are determined according to the ls-bound-consistency definition. Corollary 3 implies that there are no fixed-parameter tractable algorithm for Algorithm 1 since $hs_{ll}\langle \mathcal{C} \rangle$ is a special case for $bc_{ls}\langle \mathcal{C} \rangle$.

Algorithm 2. $bc_{ll}\langle atleast1(\{s_1, \dots, s_m\}) \rangle(X_{ls} = ls\langle l, u, n, r, p \rangle)$

- 1: $l' \leftarrow \min_{\preceq} \{s \in X_{ls} \mid \bigwedge_{1 \leq i \leq m} |s \cap s_i| \leq 1\}$
 - 2: $u' \leftarrow \max_{\preceq} \{s \in X_{ls} \mid \bigwedge_{1 \leq i \leq m} |s \cap s_i| \leq 1\}$
 - 3: **return** $ls\langle l', u', n, r, p \rangle$
-

Theorem 4. Algorithm 2 runs in time $O(n^c mc)$ where $c = |u|$.

Proof. X_{ls} contains at most $O(n^c)$ sets. Each set takes $O(mc)$ time to verify if it satisfies the constraint. \square

Sometimes enumerating all possible solutions is not cost-effective and hence we also consider an exponential filtering algorithm (Algorithm 2) for the length-lex bounds only. Obviously, lines 1–2 do not compute the set of solutions explicitly but only searches for the smallest and largest solution in the length-lex ordering. Algorithm 2 has the same worst case time complexity as Algorithm 1, since its feasibility routine is $W[1]$ -hard, but it may be significantly faster in practice. The same principles can be applied to other unary intersection constraints.

6 Experimental Results

Social Golfer Problem. Since the model uses a vanilla week-wise labeling strategy, when we label week w , all variables $X_{w',g'}$ in earlier weeks ($\forall w' < w$) are bounded. As a consequence, we can set $s_i = X_{w',g'}$ and introduce the *atmost1* constraint. We apply the following rule for all variables $X_{w,g}$:

$$\frac{\forall w' < w, g' : X_{w',g'} \text{ is bounded}}{\forall w' < w, g' : |X_{w,g} \cap X_{w',g'}| \leq 1 \mapsto atleast1(\{X_{w',g'} \mid w' < w\}, X_{w,g})}$$

Table 3 reports the experimental results on the *atmost1* propagator using instances in Table 1 and 40 larger and harder instances. The ls-domain model with $bc_{ls}\langle atleast1 \rangle$ is the only model to solve all instances within the time limit. It is the fastest model for all but one instance. On the traditional instances, the model is more than 6 times faster than the BDD-SAT-VSIDS approach. It is important to note that the search procedure is static and completely uninformed: All the reasoning is taking place in the propagation. The results also show the complementary between the exponential propagator and the richer domain, since the ll-domain, even with the exponential propagator for the length-lex component, is not as fast and robust.

The exponential propagators proposed in this paper have two roles: They increase the amount of filtering and they accelerate the convergence of the fixpoint algorithm. Table 4 shows that the effects are cumulative on the social golfer problem. The results are obtained on two difficult instances from the standard benchmarks in Table 1 (as the remaining ones are too easy) as well as from larger and more difficult instances. The results compare the ls-domain with three versions

Table 3. The Benefits of the *atmost1* Constraint on the Social Golfer Problem

	BDD-SAT -VSIDS	ls-domain		ll-domain + $bc_{ll}(atmost1)$		ls-domain + $bc_{ls}(atmost1)$	
g,s,w	Time	Time	Fails	Time	Fails	Time	Fails
58 Easy	140.83	6.49	2570	9.67	3649	5.98	1808
5,3,7	6.32	12.1	6274	17.82	7660	10.31	5373
6,5,5	15.24	21.61	9948	36.26	17245	8.42	4841
Sub-total	162.39	40.2	18792	63.75	28554	24.71	12022
5,4,6		196.2	104570	258.6	130658	102.5	56174
6,5,6		232.8	65734	352.2	107362	79.06	29578
6,6,4		×	×	×	×	1775	890645
7,3,8		0.2	12	0.21	13	0.17	8
7,4,6		0.14	5	0.15	6	0.14	20
7,5,5		0.38	96	0.57	228	0.15	11
7,6,4		0.61	196	0.66	312	0.24	58
7,7,4		272.8	44954	2.82	895	1.6	333
8,3,10		478.6	59610	1435	222609	166.2	26282
8,4,7		2.49	416	14.57	5125	0.62	111
8,5,6		11.42	2448	29.16	10166	1.07	333
8,6,5		37.34	7080	43.06	15740	2.64	1081
8,7,4		6.68	1592	10.42	4160	1.18	413
9,3,11		49.96	4254	191.7	26745	16.46	1990
9,4,8		29.75	4031	73.29	16732	3.26	496
9,5,7		414.5	46835	×	×	16.08	3462
9,6,6		×	×	×	×	55.56	17332
9,7,5		×	×	×	×	20.9	6923
9,8,4		657	97659	1164	282668	37.21	14609
10,3,12		12.48	909	×	×	2.33	61
10,4,9		20.34	1926	363.7	54956	2.56	137
10,5,7		4.21	572	79.36	14340	1.59	64
10,6,6		17.33	2386	13.39	2125	3.36	454
10,7,5		11.54	1673	115.4	21687	3.18	286
10,8,4		39.98	3457	85.17	12595	38.9	3560
11,3,13		8.19	310	135.4	14005	3.42	18
11,4,10		11.68	606	849.3	98803	2.18	12
11,5,8		628.3	62145	76.52	10199	15.09	2331
11,6,7		1497	115092	×	×	18.17	2687
11,7,5		5.15	100	3.66	144	3.21	81
11,8,3		427.5	26379	274.9	36739	187.8	30144
11,9,3		338	38996	465.5	60785	283.6	43524
11,10,3		24.09	2545	43.35	5454	22.09	3045
12,3,14		12.38	382	864.79	77133	8.85	217
12,4,11		×	×	×	×	84.9	5459
12,5,9		×	×	×	×	70.05	8639
12,6,7		132.7	19374	76.29	7403	6.25	114
12,7,6		21.91	622	×	×	20.24	424
12,8,4		39.45	2502	49.44	3019	41.18	2630

Table 4. An Analysis of the Benefits of the *Atmost1* Constraint

(g,s,w)	ls-domain		ls-domain + $bc_{ll}\langle atmost1 \rangle$		ls-domain + $bc_{sb}\langle atmost1 \rangle$		ls-domain + $bc_{ls}\langle atmost1 \rangle$	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails
5,3,7	12.1	6274	11.73	6250	10.55	5376	10.31	5373
5,4,6	196.2	104570	169.4	102385	112	56260	102.5	56174
6,5,5	21.61	9948	15.88	9536	10.52	4849	8.42	4841
6,5,6	232.8	65734	169.5	63113	97.21	29639	79.06	29578
6,6,4	×	×	1687	958106	×	×	1775	890645
9,3,11	49.96	4254	31.67	4248	20.06	1989	16.46	1990
9,4,8	29.75	4031	15.17	3994	5.09	495	3.26	496
9,5,7	414.5	46835	156.1	44247	31.77	3460	16.08	3462
9,6,6	×	×	1718	566144	121.6	17332	55.56	17332
9,7,5	×	×	×	×	59.31	6923	20.9	6923
9,8,4	657	97659	172.5	72688	107.6	14609	37.21	14609
12,3,14	12.38	382	11.75	382	9.33	217	8.85	217
12,4,11	×	×	1018	99989	128.3	5457	84.9	5459
12,5,9	×	×	×	×	97.76	8639	70.05	8639
12,6,7	132.7	19374	126.1	19370	13.08	114	6.25	114
12,7,6	21.91	622	19.77	626	50	424	20.24	424
12,8,4	39.45	2502	46.1	2631	52.76	2630	41.18	2630

of the exponential propagator: bc_{ll} which only updates the length-lex bounds, bc_{sb} which only updates the required and possible, and the complete propagator bc_{ls} . $bc_{ls}\langle atmost1 \rangle$ and $bc_{sb}\langle atmost1 \rangle$ should produce the same search tree, while $bc_{ll}\langle atmost1 \rangle$ and ls-domain should also explore the same but larger tree since these models do not have extra propagation on the required and possible sets.³ Therefore the comparison between $bc_{ls}\langle atmost1 \rangle$ and $bc_{sb}\langle atmost1 \rangle$ on the one hand and $bc_{ll}\langle atmost1 \rangle$ and ls-domain on the other hand measures how much the *atmost1* propagator speeds up the convergence of the fixpoint algorithm. The comparison between $bc_{ls}\langle atmost1 \rangle$ and $bc_{ll}\langle atmost1 \rangle$ measures the benefits from the additional pruning obtained by propagating both bounds simultaneously.

Observe first that $bc_{sb}\langle atmost1 \rangle$ can be significantly slower than $bc_{ls}\langle atmost1 \rangle$. Instance (9,8,4) is particularly interesting in that regard. $bc_{ls}\langle atmost1 \rangle$ takes about 37 seconds, while $bc_{sb}\langle atmost1 \rangle$ completes in about 107 seconds. So, on this instance, $bc_{ls}\langle atmost1 \rangle$ speeds up the propagation by a factor close to 3 by using an exponential propagator. However, the results for $bc_{ll}\langle atmost1 \rangle$ show the significant benefits of additional propagation coming from the product of the length-lex and subset-bound domains. Since both of them use the exponential propagator for the length-lex component, the benefits come from

³ There are some differences between the number of failures in our implementation since the fixpoint algorithm can halt prematurely, i.e., when the number of iterations exceeds a threshold based on the number of constraints. Similar techniques are used for continuous constraint propagation.

Table 5. The Benefits of the *exact1* Constraint on the Steiner Triple System

n	ls-domain		ls-domain + $bc_{ll}\langle exact1 \rangle$		ls-domain + $bc_{ls}\langle exact1 \rangle$	
	Time	Fails	Time	Fails	Time	Fails
7	0.01	0	0.01	0	0.01	0
9	0.01	1	0.01	1	0.01	1
13	0.02	51	0.01	1	0.02	1
15	0.04	73	0.04	1	0.04	1
19	0.17	140	0.15	0	0.19	0
21	0.35	192	0.26	0	0.40	0
25	5.63	912	0.55	0	2.66	0
27	12.61	1826	0.89	0	4.74	0
31	2.03	0	2.22	0	2.95	0
33	24.32	1714	4.33	1	19.47	0
37	1345.76	16178	24.24	0	1443.32	0
39	505.57	5248	21.92	0	1307.45	0
43	×	×	403.20	0	×	×
45	×	×	1344.79	0	×	×

the additional filtering, not the speed of the fixpoint algorithm. The experimental results show significant improvements in efficiency in favor of the richer domain. For instance, $bc_{sb}\langle atleast1 \rangle$ terminates after 20 and 70 seconds on instances (9, 7, 5) and (12, 5, 9), while $bc_{ll}\langle atleast1 \rangle$ does not terminate after 1,800 seconds.

Steiner Triple System. The model uses a static smallest index variable first labeling strategy. When we label variable X_i , all lower indexed variables $X_{i'}$ ($\forall i' < i$) are bounded. We set $s_{i'} = X_{i'}$ and introduce the *exact1* constraint.

$$\frac{\forall i' < i : X_{i'} \text{ is bounded}}{\forall i' < i, |X_i \cap X_{i'}| = 1 \mapsto exact1(\{X_{i'} \mid i' < i\}, X_i)}$$

Table 5 reports the experiment results on the *exact1* propagator using the instances in Table 2. Here the ls-domain with the length-lex bound propagator ($bc_{ll}\langle exact1 \rangle$) gives the best results and improves the state-of-the-art considerably. It solves two more instances that are unsolvable by the original model and is several orders of magnitude faster than the original model and the BDD-SAT formulations. *What is particularly remarkable is that the model has almost no failure on these instances and that the only benefit of the exponential propagator is to speed up the computation of the fixpoint algorithm.* The complete filtering algorithm ($bc_{ls}\langle exact1 \rangle$) is much slower since it must explore a considerable number of sets. The additional pruning that it brings is not compensated by the increased computational costs. What this benchmarks shows is that speeding up the fixpoint algorithm by exponential propagators may produce substantial improvements in efficiency.

7 Related Work

This section briefly reviews some related work on exponential propagation and propagators. Perhaps the closest related work is the work on box consistency in the Numerica system [23]. The key idea of box consistency was to avoid the decomposition of a complex constraints into elementary ternary constraints. By enforcing box consistency on the original constraint, these systems improve the pruning, addresses the so-called dependency effect of interval propagation, and tackle the fact that the fixpoint algorithm can take a long time to converge. Box consistency was enforced by a potentially exponential algorithm. The Newton and Numerica systems also include conditions to terminate the fixpoint algorithm prematurely when the propagation was not reducing the search space enough. Lebbah and Lhomme [12] considered the use of extrapolation methods to speed up the convergence of filtering algorithms for continuous CSPs, also dramatically the efficiency on these problems. These techniques could potentially be applied to set domains as well, but this paper took another, simpler, route: Using exponential propagators that have a more global view of the problem at hand. Also closely related is the work of Bessière and Régis on solving CSPs on the fly. They recognize that, on certain applications, the pruning offered by the solver was not strong enough. They isolated a global constraint (i.e., the sum of n variables taking different values) for which they did not design a specific propagator. Instead, they use the CP solver recursively and solved CSPs on the fly to enforce arc consistency. Once again, the result is to move some of the exponential behavior from the search to the constraint propagation. Note also that several pseudo-polynomial algorithms have also been proposed in the past, including the well-known filtering algorithm for knapsack constraints [24].

8 Conclusion

Most research in constraint programming focuses on designing polynomial-time filtering algorithms. This paper explored, for set CSPs, the idea of shifting some of the exponential behavior from the search component to the filtering component, and from the constraint-propagation algorithm to the propagators. It showed that the product of the length-lex and subset-bound domains improves state-of-the-art results significantly despite a potential exponential propagation algorithm. More importantly, it presented exponential-time propagators for intractable unary intersection constraints and demonstrated that they bring considerable performance improvement by speeding up constraint propagation and increasing filtering. They indicate that it may sometimes be beneficial to embrace complexity in the filtering component and exploit the constraint semantics and locality, instead of relying on rather agnostic search and constraint propagation algorithms.

References

1. Bessière, C., Régis, J.C.: Enforcing arc consistency on global constraints by solving subproblems on the fly. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 103–117. Springer, Heidelberg (1999)

2. Puget, J.F.: Pecos a high level constraint programming language. In: Proc. of Spicis (1992)
3. Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints* 1(3), 191–244 (1997)
4. Sadler, A., Gervet, C.: Global reasoning on sets. In: Proceedings of Workshop on Modelling and Problem Formulation (FORMUL 2001), held alongside CP-2001 (2001)
5. Azevedo, F.: Cardinal: A finite sets constraint solver. *Constraints* 12(1), 93–129 (2007)
6. Bessière, C., Hebrard, E., Hnich, B., Walsh, T.: The complexity of global constraints. In: McGuinness, D.L., Ferguson, G. (eds.) AAAI, pp. 112–117 (2004)
7. Peter Hawkins, V.L., Stuckey, P.J.: Solving set constraint satisfaction problems using robdds. *Journal of Artificial Intelligence Research* 24, 109–156 (2005)
8. Gervet, C., Van Hentenryck, P.: Length-lex ordering for set csp. In: AAAI (2006)
9. Sellmann, M.: On decomposing knapsack constraints for length-lex bounds consistency. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 762–770. Springer, Heidelberg (2009)
10. Sadler, A., Gervet, C.: Hybrid set domains to strengthen constraint propagation and reduce symmetries. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 604–618. Springer, Heidelberg (2004)
11. Malitsky, Y., Sellmann, M., Van Hoes, W.: Length-Lex Bounds Consistency for Knapsack Constraints. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 266–281. Springer, Heidelberg (2008)
12. Lebbah, Y., Lhomme, O.: Accelerating filtering techniques for numeric csp. *Artif. Intell.* 139(1), 109–132 (2002)
13. Mercier, L., Van Hentenryck, P.: Edge finding for cumulative scheduling. *INFORMS Journal on Computing* 20(1), 143–153 (2008)
14. Yip, J., Van Hentenryck, P.: Evaluation of length-lex set variables. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 817–832. Springer, Heidelberg (2009)
15. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness ii: On completeness for $w[1]$. *Theor. Comput. Sci.* 141(1&2) (1995) 109–131
16. Gange, G., Lagoon, V., Stuckey, P.: Fast set bounds propagation using a bdd-sat hybrid. To appear in JAIR (2010)
17. Van Hentenryck, P., Yip, J., Gervet, C., Dooms, G.: Bound consistency for binary length-lex set constraints. In: Fox, D., Gomes, C.P. (eds.) AAAI, pp. 375–380 (2008)
18. van Hoes, W.J., Sabharwal, A.: Filtering atmost1 on pairs of set variables. In: Peron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 382–386. Springer, Heidelberg (2008)
19. Yip, J., Van Hentenryck, P., Gervet, C.: Boosting set constraint propagation for network design. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 339–353. Springer, Heidelberg (2010)
20. Barnier, N., Brisset, P.: Solving the kirkman’s schoolgirl problem in a few seconds. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 477–491. Springer, Heidelberg (2002)
21. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver, pp. 530–535 (2001)
22. Colbourn, C.J., Dinitz, J.H., Li, L.C., Jajcay, R., Magliveras, S.S. (eds.): *The crc handbook of combinatorial designs* (1995)
23. Van Hentenryck, P., Michel, L., Deville, Y.: *Numerica: a Modeling Language for Global Optimization*. The MIT Press, Cambridge (1997)
24. Trick, M.A.: A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals OR* 118(1-4), 73–84 (2003)

An Empirical Study of Optimization for Maximizing Diffusion in Networks*

Kiyan Ahmadizadeh, Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal

Department of Computer Science, Cornell University, Ithaca NY 14853, USA
{kiyan,bistra,gomes,sabhar}@cs.cornell.edu

Abstract. We study the problem of maximizing the amount of stochastic diffusion in a network by acquiring nodes within a certain limited budget. We use a Sample Average Approximation (SAA) scheme to translate this stochastic problem into a simulation-based deterministic optimization problem, and present a detailed empirical study of three variants of the problem: where all purchases are made upfront, where the budget is split but one still commits to purchases from the outset, and where one has the ability to observe the stochastic outcome of the first stage in order to “re-plan” for the second stage. We apply this to a Red Cockaded Woodpecker conservation problem. Our results show interesting runtime distributions and objective value patterns, as well as a delicate trade-off between spending all budget upfront vs. saving part of it for later.

1 Introduction

Many real-world processes are diffusive in nature, giving rise to optimization problems where the goal is to maximize or minimize the spread of some entity through a network. For example, in epidemiology, the spread of infectious diseases in a human or animal network is a diffusion-based process. In ecology, so-called metapopulation models capture the diffusion of species in a fragmented landscape of habitat patches. Similarly, the adoption of a certain marketed product by an individual may trigger his or her friends or fans to adopt that product as well, suggesting viral marketing strategies in human networks. In the social network setting, particularly in Internet-based networks such as Facebook and Twitter, the spread of information between individuals is yet another diffusion process. The stochastic nature of such diffusion processes, or *cascades*, and how best to intervene in order to influence their outcomes, has been the study of several recent papers in these areas [e.g. 2, 4, 6, 8, 10, 11, 13]. A key question in this context is, if one had limited resources to purchase part of the network to use either as the initially “active” nodes or as nodes that may participate in the diffusion process, which nodes should one purchase?

We study this question with a focus on the case where the intervention budget, instead of all being available upfront, is split into two or more time steps.

* Supported by NSF (Expeditions in Computing award for Computational Sustainability, 0832782; IIS grant 0514429) & AFOSR (IISI, grant FA9550-04-1-0151). The authors thank Yahoo! for generously providing access to their M45 compute cloud.

We evaluate our techniques on a specific conservation problem in which the diffusion process of interest is the dispersal and territory establishment of an endangered bird species, the Red-Cockaded Woodpecker (RCW) [3, 5, 15]. Using a stochastic model of the diffusion of RCW through geographic maps, we study the solutions of three optimization variants that differ in when the conservationist is allowed to make purchase decisions. The methodology and results apply not only to RCW conservation but also to many other problems in the field of *computational sustainability* [7], where maximizing or minimizing diffusion is a commonly encountered problem in areas ranging from ecology to poverty mitigation through food networks.

Formally, we pose this optimization task as a stochastic Mixed Integer Programming (MIP) problem. Even extremely simple classes of stochastic linear programs are #P-hard [cf. 15]. However, an effective solution method with stochastic optimality gap guarantees is the Sample Average Approximation (SAA) [14]. The application of SAA to a formulation of the RCW conservation problem has been previously evaluated on a real-life size instance, motivated by the actual data for the RCW population in North Carolina [15]. In order to better understand the effectiveness and scalability of this solution methodology from a computational perspective, we introduce a synthetic problem generator¹ that uses real map data as a basis for generating random instances. This analysis reveals several interesting computational trends: an easy-hard-easy runtime pattern as the budget fraction is varied, an increase in hardness as the “self-colonization” probability is increased, more runtime variation for instances that are harder to solve (for a fixed budget), and a roughly inverse relation between the computational hardness and the solution objective value (again for a fixed budget).

We also study a natural and realistic generalization of the problem where the total budget is actually not available to be spent at the beginning but is rather *split* into two stages. This modification significantly increases the computational difficulty of the problem and makes it non-trivial to apply the SAA methodology which was originally designed for the case where stochastic samples can be drawn right upfront, rather than adaptively. Indeed, the most general approach, a truly *multi-stage stochastic version* of the problem [1], has very limited scalability. Instead, we consider two simpler problem variants that are more scalable: committing to both first and second time step purchase decisions upfront (the *single-stage method for the split-budget problem*) or committing to purchase decisions for the first time step but re-evaluating what is best to purchase in the second time step after observing the stochastic outcome of the first stage (the *two-stage re-planning method*). Our experiments show that the re-planning approach, although computationally more expensive, does pay off—the solution objective obtained through an SAA-style implementation of re-planning is often significantly better than that obtained by the single-stage split-budget with all purchase decisions made upfront; more interestingly, the value of information gained from first stage observations can result in a re-planning objective that is better than spending all budget upfront.

¹ <http://www.cs.cornell.edu/~kiyan/rcw/generator.htm>

2 Problem Description, Model, and Solution Methods

A diffusion process can be quite complex; *patch-based models* [8] represent diffusion as occurring on a network of *nodes* which become *active* upon the arrival of the dispersing entity. Given a network of nodes (i.e., an undirected graph), a *network dispersion model* specifies for each pair of nodes (r_1, r_2) the probability that node r_2 will become active at the next time step, given that r_1 is currently active. Similarly, if node r is currently active, it remains so after one time step with a specified *survival probability*. In our optimization setting, we assume that the process can only spread to nodes that have been purchased. We divide the nodes into disjoint *node sets* with an associated cost and assume that a manager making purchase decisions is limited by a budget in each time step. Given the stochastic nature of the dispersion process, the overall goal is to maximize the *expected* number of active nodes at a specified *time horizon*.

For our experiments, we consider a specific instance of this problem in which the diffusing entity is the Red-Cockaded Woodpecker (RCW). Geographic territories suitable for RCW inhabitation represent graph nodes, with territories grouped into real estate parcels available for purchase. Node activity here represents the settlement of a territory by RCW members dispersing from other territories. As in most conservation settings, the geographic territories must be owned and maintained by conservationists for species members to survive there.

We formulate this problem as a stochastic Mixed Integer Program (MIP), shown below. (One can create alternative MIP formulations of this problem as well, using, e.g., network flow.) Let R be the number of nodes in the network, P the number of node sets, H the planning horizon, $C(p)$ the cost of node set $p \in \{1..P\}$, $B(t)$ the budget available at time step $t \in \{0..H-1\}$, $P(r)$ the node set that node $r \in \{1..R\}$ belongs to, and $I(r)$ the 0-1 indicator of whether node r is initially active (node sets containing an initially active node are assumed to be already owned). Binary variables $\{y(p, t) \mid p \in \{1..P\}, t \in \{0..H-1\}\}$ correspond to the action of buying node set p at time step t . Binary variables $\{x(r, t) \mid r \in \{1..R\}, t \in \{0..H-1\}\}$ correspond to r being active at time t . The constraints of the MIP encode the basic requirements discussed earlier in the problem description. For lack of space, we refer the reader to [15] for details and mention here only that the budget constraint (2) has been generalized to include a time-step-specific budget and that $\xi_{r', r}^{t-1}$ are the stochastic coefficients that follow the dispersion model probability for r' and r .

In reality, we cannot directly optimize this stochastic MIP. Instead, we use the Sample Average Approximation (SAA) method [14, 18], which uses random samples from the underlying probability distribution of the stochastic parameters to generate a finite number of scenarios and creates a deterministic MIP to optimize the *empirical average* (rather than the true expectation) of the number of active territories over this finite set of sampled scenarios. We will describe this shortly. In this deterministic version of our stochastic MIP defined over a set of k scenarios S_1, S_2, \dots, S_k , we still have one purchase variable for each node set at each time step but k different activity variables for each node at each time step, capturing the different diffusion activity in the k different scenarios. In other

$\text{maximize } \sum_{r=0}^R x(r, H) \quad \text{such that}$	
$y(p, 0) = 1$	$\forall p \in \text{initial (free) parcels} \quad (1)$
$\sum_{p=1}^P C(p) \times y(p, t) \leq B(t)$	$\forall t \in \{0..H - 1\} \quad (2)$
$\sum_{t=0}^{H-1} y(p, t) \leq 1$	$\forall p \in \{1..P\} \quad (3)$
$x(r, t) \leq \sum_{t'=0}^t y(P(r), t')$	$\forall r \in \{1..R\}, \forall t \in \{1..H\} \quad (4)$
$x(r, t) \leq \sum_{r'=1}^R \xi_{r',r}^{t-1} x(r', t - 1)$	$\forall r \in \{1..R\}, \forall t \in \{1..H\} \quad (5)$
$x(r, 0) = I(r)$	$\forall r \in \{1..R\} \quad (6)$

words, the purchase decisions are synchronized amongst the different scenarios but activity variables differ depending on which nodes were occupied in which scenario. For the objective function, we simply sum up the activity variables at the horizon for all k scenarios, thus optimizing the sum of active territories (or, equivalently, the average activity) over the k scenarios at the horizon. This results in an expanded deterministic formulation very similar to the one above, and we denote it by $\text{MIP}(S_1, S_2, \dots, S_k)$ [\[2\]](#)

While our MIP allows a budget constraint for each time step, in our experiments we consider two variants. In the *upfront* variant, all budget is spent at time step $T = 0$. In the *split* variant, the budget is split into (b_1, b_2) in a given proportion between $T_1 = 0$ and $T_2 < H$, and is set to 0 for other time steps.

2.1 Sample Average Approximation and Re-planning

The SAA approach has been instrumental in addressing large-scale stochastic optimization problems [\[14, 18\]](#). It provides provable convergence guarantees—it may over-fit for a small number of scenarios, but converges to the true optimum with increasing training samples and provides a statistical bound on the optimality gap. The SAA procedure works in three phases. In the TRAINING phase, we generate N candidate solutions by creating N SAA MIPs with k training scenarios each and solving them to optimality. In the VALIDATION phase, we generate M_1 new validation scenarios, evaluate each of the N candidate solutions on these scenarios, and choose the solution s^* with the best validation objective. In the TEST phase, we generate M_2 fresh scenarios to re-evaluate s^* , thus obtaining and reporting an estimate of the true objective value of s^* .

² In the deterministic MIP, we add redundant constraints that force any variable $x(r, t)$ of a scenario to be set to 1 whenever the corresponding node set has been bought and there was dispersal from nodes active at the previous time step.

The test objective of s^* is a lower bound on the true optimum while the average of the MIP objective of all N candidate solutions is a (stochastic) upper bound on the optimum (see [15] for more details). The above procedure is applied to both the upfront and split budget models. For our experiments we set $N = 100, k = 10, M_1 = 1000, M_2 = 5000$.

In the split budget setting, it is quite pessimistic to assume that decision makers cannot adjust purchase decisions based on first stage observations. The true objective evaluation of a split budget purchase plan needs to be more “dynamic” in nature—at the second decision time step T_2 one can observe the events of the past time steps and accordingly re-plan how to spend b_2 . Given a set of purchase decisions for T_1 , we describe *how to evaluate the expected objective value under re-planning*, assuming that at the second decision point T_2 , one would again apply the SAA solution method to select purchase decisions. The re-planning evaluation of a candidate solution s , representing purchase decisions made at T_1 in the split budget model, is done as follows. We generate a sample set of F “prefix scenarios” over the years $0..T_2 - 1$. For each prefix scenario and considering all nodes sets purchased in s as being available for free at T_2 , we perform an SAA evaluation as if we are at time step T_2 and are solving the *upfront* model for the remaining years and with budget b_2 . The SAA here is performed for $N = 20, k = 10, M_1 = 100$ and $M_2 = 500$, for each of $F = 100$ prefix scenarios. Finally, the re-planning objective of s is reported as the average SAA objective over the $F = 100$ prefix scenarios.

3 Experimental Results

We use a graph of nodes derived from a topology of 411 territories grouped into 146 parcels, representative of a region on the coast of North Carolina of interest to The Conservation Fund for RCW preservation. The dispersion model used for this study is based on a habitat *suitability score* (an integer in $[0, 9]$) for each territory as well as known parameters about the ability of birds to disperse between territories at various distances [12]. Suitability scores were estimated using GIS data from the 2001 USGS National Land Cover Dataset (NLCD) [16]. Parcels (corresponding to node sets) were constructed using the US Census Bureau’s “census block” divisions [17]. Using the base topology, we created several randomized instances of the problem by (a) perturbing the suitability value by ± 1 and (b) selecting different sets of initially active territories by randomly choosing clusters of territories with high suitability.

We used Yahoo!’s M45 cloud computing platform running Apache Hadoop version 0.20.1 to perform independent parts of our solution methods massively in parallel. IBM ILOG CPLEX v12.1 [9] was used to solve all MIPs involved.

Runtime Distributions and Objective Value of MIP. We study the runtime to solve the SAA MIPs (with $k = 10$ scenarios over $H = 20$ years) under different budgets expressed as a fraction of the total cost of all parcels in the instance. Results are presented in Fig. 1. Each point in these plots corresponds to the average runtime

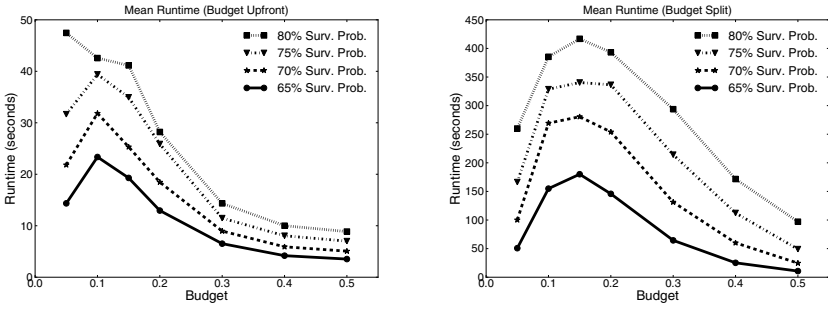


Fig. 1. Runtime (y-axis) as a function of budget (x-axis) for various extinction rates. Left: all budget available upfront. Right: budget split into two time steps.

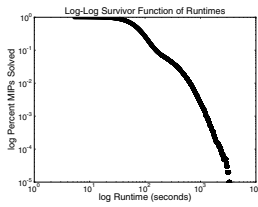


Fig. 2. Runtime distribution for instance map4-30714 exhibits power-law decay (log-log scale)

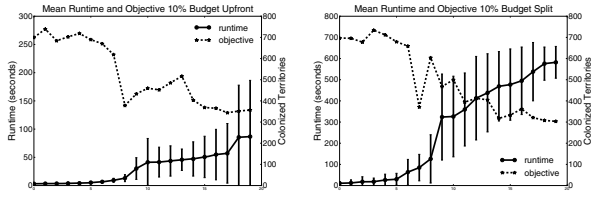


Fig. 3. The high variation in runtime on some instances (lower curve) and the corresponding average MIP objective values (higher curve)

over 100 different samples of $k = 10$ scenarios of each of 20 different variations of the basic map described earlier. The left pane shows the results when all budget is available upfront, while the right pane considers the split-budget case. These curves demonstrate an easy-hard-easy pattern as the budget parameter is varied, and also indicate that the problem becomes harder to solve for higher survival rates. Comparing the left and right plots, we see that the split-budget variant of the problem is roughly 10x harder to solve than when all budget is available upfront (notice the scales on the y-axis).

We evaluate in more detail the performance with 70% survival rate. Fig. 2 shows the distribution of the runtime for one particular variation of the base map, called map4-30714, for 10% budget. All budget is available upfront and the plot is derived from 100,000 runs. This figure demonstrates the typical runtime distribution seen on this suite of instances: a *power-law decay*, indicated by the near-linear (or super-linear) drop in the probability of “failure” or timeout (y-axis) as a function of the runtime (x-axis) when plotted in log-log scale.

We next consider the relation between the running time and the objective value of the SAA MIP, for both the upfront and split budget cases. The lower curves in the plots of Fig. 3 show the average runtime and standard deviation over 100 runs of each of 20 variations of the base map, where the 20 instances

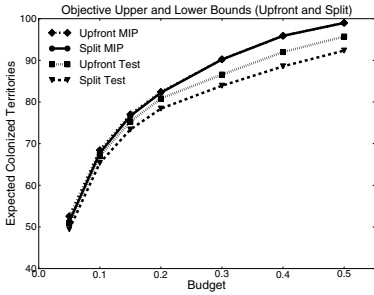


Fig. 4. SAA upper and lower bounds on obj. value for upfront and split budgets

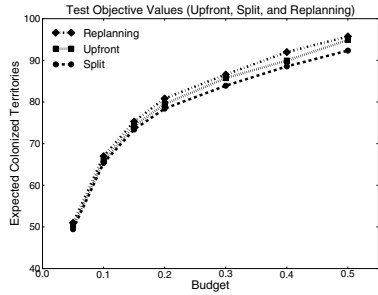


Fig. 5. Objective value of re-planning, compared to upfront and split budgets

are ordered from low to high runtime. The upper curves show the corresponding average objective value achieved for each instance (the variation in the objective value was small). These plots indicate that for our test suite, instances that are hard to solve often have a significantly higher runtime variation than instances that are easy to solve. Moreover, the harder to solve instances typically result in a lower objective value.

Evaluation of Sample Average Approximation and Re-Planning. We evaluate the solution quality of the SAA approach as a function of the budget fraction. Fig. 4 presents results for both the upfront and split budget problems where the budget is divided evenly between $T_1 = 1$ and $T_2 = 10$. The curves marked Upfront MIP and Split MIP present the average MIP objective over the $N = 100$ candidate solutions and are hence a stochastic upper bound on the true optimum. The curves marked Upfront Test and Split Test are the estimated true quality of the solution chosen (and hence provide a lower bound on the quality of the true optimum). The difference between Upfront Test and Split Test measures the penalty of not having all funds available in the first stage. The relatively small gap between the upper and lower bounds confirms that our choice of SAA parameters is good and that the solutions provided are very close to optimal.

Finally, we evaluate the advantage of re-planning in the stochastic setting of our problem. Recall that we would like to understand the tradeoff between spending all available budget upfront vs. re-planning with a portion of investments at a later stage after making stochastic observations. The balance is, in fact, quite delicate. By spending too much money upfront, we leave little room for “adjusting” to the stochastic outcome of the first stage. On the other hand investing too little upfront limits the amount of possible variation in dispersion, thus limiting the worth of stochastic observations. When splitting the budget evenly, making second stage decisions at $T_2 = 10$, re-planning often did not yield as good a result as investing all money upfront. Nonetheless, for other parameters such as a 30-70 split with $T_2 = 5$, we found that re-planning begins to pay off, as is shown in Fig. 5. The top curve in the plot corresponds to re-planning and shows that it

can result in the occupation of more territories in our bird conservation example than spending all budget upfront (the middle curve) or splitting the budget but providing a single-stage style solution that commits to a certain set of purchase decisions at the outset (the lowest curve).

In summary, our experiments have examined the complexity of optimizing stochastic diffusion processes and the value of different planning methodologies. Our results show the considerable benefits of making decisions upfront (e.g. in a single-stage), and the benefits that re-planning based on stochastic observations can have when decisions must be made in multiple stages.

References

- [1] S. Ahmed. Introduction to stochastic integer programming. *COSP Stochastic Programming Introduction* – <http://stoprog.org>, 2004.
- [2] Anderson, R., May, R.: Infectious diseases of humans: dynamics and control. Oxford University Press, Oxford (1992)
- [3] Conner, R., Rudolph, D., Walters, J., James, F.: The Red-cockaded Woodpecker: surviving in a fire-maintained ecosystem. Univ. of Texas Press (2001)
- [4] Domingos, P., Richardson, M.: Mining the network value of customers. In: KDD, pp. 57–66 (2001), ISBN 1-58113-391-X
- [5] US Fish and Wildlife Service. Red-cockaded woodpecker recovery plan (2003)
- [6] Goldenberg, J., Libai, B., Muller, E.: Talk of the network: A complex systems look at word-of-mouth. *Marketing Letters* 12(3), 211–223 (2001)
- [7] Gomes, C.P.: Computational Sustainability: Computational methods for a sustainable environment, economy, and society. *The Bridge*, NAE 39(4) (2009)
- [8] Hanski, I.: Metapopulation ecology. Oxford University Press, USA (1999)
- [9] IBM ILOG, SA. CPLEX 12.1 Reference Manual (2009)
- [10] Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: KDD, pp. 137–146 (2003)
- [11] Leskovec, J., Adamic, L., Huberman, B.: The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)* 1(1), 5 (2007)
- [12] Letcher, B.H., Priddy, J.A., Walters, J.R., Crowder, L.B.: An individual-based, spatially-explicit simulation model of the population dynamics of the endangered red-cockaded woodpecker, *picoides borealis*. *Biol. Conserv.* 86(1), 1–14 (1998)
- [13] McDonald-Madden, E., Baxter, P.W., Possingham, H.P.: Making robust decisions for conservation with restricted money and knowledge. *Appl. Ecol.* 45(9), 1630–1638 (2008)
- [14] Shapiro, A.: Monte Carlo sampling methods. In: Ruszczyński, A., Shapiro, A. (eds.) *Stochastic Programming. Handbooks in Operations Research and Management Science*, vol. 10, pp. 353–426 (2003)
- [15] Sheldon, D., Dilkina, B., Elmachtoub, A., Finseth, R., Sabharwal, A., Conrad, J., Gomes, C.P., Shmoys, D., Allen, W., Amundsen, O., Vaughan, B.: Optimal network design for the spread of cascades. Technical report, Cornell University (April 2010), <http://hdl.handle.net/1813/14917>
- [16] The USGS Land Cover Institute (LCI). USGS land cover (2001)
- [17] US Census Bureau. Census data: 2009 TIGER/Line shapefiles (2009)
- [18] Verweij, B., Ahmed, S., Kleywegt, A., Nemhauser, G., Shapiro, A.: The sample average approximation method applied to stochastic routing problems: a computational study. *Computational Optimiz. and Applications* 24(2), 289–333 (2003)

An Integrated Modelling, Debugging, and Visualisation Environment for G12

Andreas Bauer^{1,2}, Viorica Botea¹, Mark Brown¹, Matt Gray^{1,2},
Daniel Harabor^{1,2}, and John Slaney^{1,2}

¹ National ICT Australia (NICTA)*

² The Australian National University

Abstract. We present G12IDE, a front-end for the G12 platform aimed at helping users create and work with constraint models in a manner independent from any underlying solver. G12IDE contains tools for writing and evaluating models using Zinc and provides a feature rich debugger for monitoring a running search process. Debugging a search, as opposed to debugging sequential code, requires concepts such as breakpoints and queries to be applied at a higher level than in standard debuggers. Our solution is to let users define special events which, once reached in a search, cause the debugger to halt and give back, possibly in a visual manner, useful information on the current state of the search. G12IDE also includes a number of visualisation tools for drawing graphs and trees, and additionally allows users to create arbitrary domain-specific visualisations, such as the drawing of a sequential plan when the constraint problem is in fact a planning problem. The inclusion of such powerful and flexible visualisation toolkit and its tight integration with the available debugging facilities is, to the best of our knowledge, completely novel.

1 Introduction

G12 [15] is a software platform for solving combinatorial optimisation problems. It supports linear and mixed integer programming, constraint propagation and inference and a variety of other search and inference-based approaches for solving complex problems. Like several other modern modelling languages [5,6], it separates the “conceptual” or constraint model from the constraint program. In G12, the constraint model is written in Zinc, a purely declarative language that can be mapped to a range of lower-level models, and ultimately to constraint programs, which may solve the problem in quite different ways. The clear separation of modelling from solving requires a shift from thinking about problem solving in terms of programs and execution to thinking in terms of models and search. This change in paradigm calls for new tools that directly support working at such a high level.

In this paper we present G12IDE, a novel integrated modelling, debugging and visualisation environment which has been developed largely in parallel to the rest of G12.

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

Built on top of the Eclipse platform,¹ G12IDE has a similar look-and-feel to a classical programming environment. It offers support for the modelling of constraint problems via an integrated Zinc editor, performs automatic builds and allows step-wise debugging of an active search process.

Our debugging system is built around an explicitly defined (and solver agnostic) schema which specifies a wide range of “interesting” search events that a user may “subscribe” to. For example, it is possible to pause the search when a variable has become grounded, when the search has reached a fixpoint or when a solution has been found. Furthermore, existing search events can be combined together to create new custom search events. This allows for a high degree of control over the search process at varying levels of granularity.

Our environment also allows constraint problems to be visualised. Users can choose between pre-defined visualisations, such as a constraint graph and search tree, or alternatively may define their own. The latter is facilitated by drawing objects in a simple graphics editor and animating them using a dedicated scripting language. In this way, not only are generic views (trees, graphs) available, but it is also possible to visualise problems in a more domain-specific manner. For example, a planning problem can be visualised by drawing the execution of the plan so far as well as the possible choices in the current search node. The inclusion of such a powerful and flexible visualisation tool in the IDE, and its tight integration with the available debugging facilities, is to the best of our knowledge, completely novel.

Outline. The rest of this paper is organised as follows. In the next section, we provide more conceptual details on debugging a running search, outlining the differences from classical debugging and the specific challenges of our domain. In Sec. 3, we give a brief architectural overview on the G12IDE, and explain its main components, or layers. The modelling layer is explained in greater detail in Sec. 4, and the visualisation layer in Sec. 5. The technical realisation of our debugging layer, or rather its relation to the underlying constraint solver, is explained in Sec. 6. Finally, Sec. 7 contains references to related work, while a brief summary and conclusions of our paper are to be found in Sec. 8.

2 Debugging Search

Bugs—errors or infelicities—may exist anywhere in the software system. They may afflict the model, the data, the mapping down to solvers or the underlying programs. Bugs in the code of solvers are not our present concern: we assume the constraint program and its associated constraint solvers work perfectly unless forced to conclude otherwise. Instead, we focus on bugs that may arise during the development of a constraint model.

Errors in the model may call for *correctness debugging*, if they affect semantics by allowing unintended solutions or by excluding intended ones. It is also common for models to contain logically correct but poorly expressed constraints. Such situations can frequently limit the effects of propagation and so *performance debugging* of the model may also be required. We thus require debugging tools which detect either static

¹ <http://www.eclipse.org/>

features of constraint models or dynamic features of the search process, and report them to us in a form appropriate to the high level at which we wish to think about constraint problems.

In a sense, this is a radical departure, but in another sense it is little different from the concept of debugging at other levels. Consider a debugger for C programs, for instance. It works with an ontology appropriate to programming at that level. It allows breakpoints to be set on lines of the C code or on C functions, not on assembly-level instructions, and when stepping to the next instruction, it breaks on the next C line rather than on the next machine instruction or the next clock cycle. When it reports the values of variables, these are variables declared in C, not the contents of registers and accumulators. Moving up from the level of program and execution to that of model and search is more of the same. Breakpoints and steps for our purposes should make sense in terms of search: “step to the next node of the search tree” and “break whenever propagation reaches a fixpoint” belong at this level, whereas “step over the next function call in the SAT solver” does not. Similarly, when we ask for the values of variables, we want to know the domains of decision variables declared in the Zinc model, not the details of whatever data structures these have turned into after mapping to solvers.

The task of our debugger is to monitor the search process, which requires it to place breakpoints in low-level code in order to collect information with which it can maintain models of the current search state, and then to pass just the right information, on demand, to the front-end tools which display it. Managing this in a systematic way seems to be new in constraint programming, so we have had to design an architecture for the search debugger at the same time as experimenting with modes of visual presentation in order to present abstractions of the search states which are likely to be useful. Details of our design decisions and the resulting tools are presented in the next sections.

3 Architectural Overview

Our environment can be described in terms of a three-tier architecture comprising a Modelling Layer, a Solving Layer and a Visualisation Layer. Fig. 1 illustrates this idea.

The Modelling Layer is where most of the interaction with the user takes place. It comprises two components: The first is a dedicated code editor for Zinc which offers features such as syntax checking, syntax highlighting, and standard Eclipse functionality such as project-based code separation. The second component is a Visualisation Editor which includes a simple canvas for drawing objects and an associated text editor for writing animation scripts. The idea is to create a custom visualisation by drawing objects on the canvas and then writing a script to define their behaviour in response to specified search events (e.g. changing the colour or position of an object in response to a variable becoming grounded). Sec. 4 describes the components of the Modelling Layer in more detail.

The Solving Layer comprises the main interface to the rest of the G12 platform (i.e., the different solvers) and our environment. Given a Zinc model, the Solving Layer is responsible for invoking the solvers, maintaining communication between the solver monitor and the debugging interface and is responsible for sending any updates regarding the state of the running search process to the appropriate visualisers. Sec. 6 contains a detailed discussion of the different components in the Solving Layer.

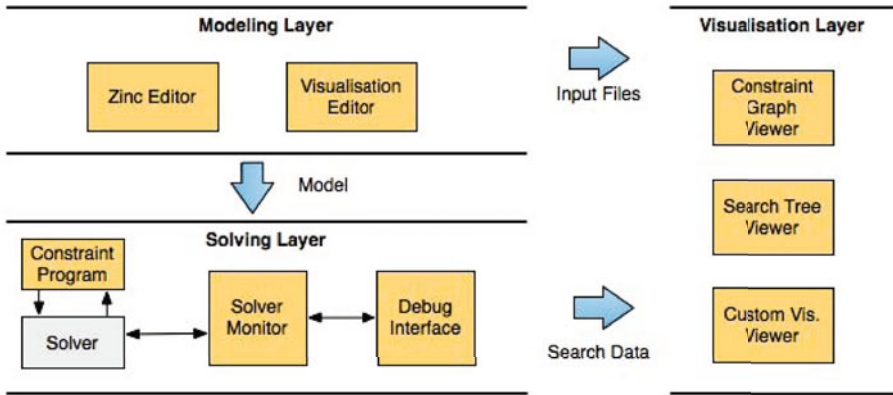


Fig. 1. The G12IDE architecture. Most features can be categorised into one of three distinct layers: modelling, solving or visualisation.

The Visualisation Layer is comprised of three dedicated visualisers: a Constraint Graph Viewer (CGV), a Search Tree Viewer (STV) and a Custom Visualisation Viewer (CVV). The first two are highly specialised, offering a range of features specific to the display of graphs and trees. The CVV on the other hand is extremely generic; while intended principally for 2-dimensional diagrams, it can display arbitrary graphics. Its applicability is thus limited only by the constraint programmer’s imagination. Each visualiser requires certain input files to function. In the case of CGV and STV the model itself suffices. The CVV also requires a visualisation script. For details on these viewers as well as the scripting language, see Sec. 5.

4 The Modelling Layer

Two components make up the Modelling Layer: a Zinc Editor and a Visualisation Editor. To better illustrate these tools and their use, we shall refer to the *meet-pass problem* as a running example. This is a standard (if not very difficult) benchmark problem in AI planning, described as follows:

The Meet-Pass Problem

Five sectors of railway track, S_1, \dots, S_5 , are linearly connected. There is a siding accessible from S_3 big enough to hold one train. Initially, there are trains in sectors S_1, S_2 and S_4 . The safety rules are that no two trains may be in the same sector at the same time, and no train may enter a sector occupied by another train, even if that other train is about to move on. Trains may only move to adjacent sectors, of course. Find the shortest plan that moves the train on S_1 to S_5 and returns the other two trains to their starting positions.

While this is a toy example, the problem class from which it comes is real enough: meet-pass planning is a constant issue in scheduling train movements. Fig. 2 shows a

```

set of int: Sectors = 1..6;
set of int: Trains = 1..3;
set of int: Steps = 1..13;
array[Sectors,Sectors] of bool: linked;
array[Trains] of Sectors: start;
array[Trains] of Sectors: finish;
array[Steps,Trains] of var Sectors: pos :: is_output;

constraint forall(t1, t2 in Trains, x in Steps)
  (pos[x,t1] == pos[x,t2] -> t1 == t2);
constraint forall(t in Trains)
  (pos[1,t] == start[t] /\ pos[nsteps,t] == finish[t]);
constraint forall(t in Trains, x in Steps where x > 1)
  (linked[pos[x,t],pos[x-1,t]]);
constraint forall(t,u in Trains where t != u)
  (forall(x in Steps where x > 1)(pos[x-1,u] != pos[x,t]));

start = [ 1, 2, 4 ];
finish = [ 5, 2, 4 ];
linked = [| true,  true,  false, false, false, false
          | true,  true,  true,  false, false, false
          | false, true,  true,  true,  false, true
          | false, false, true,  true,  true,  false
          | false, false, false, true,  true,  false
          | false, false, true,  false, false, true |];

solve satisfy;

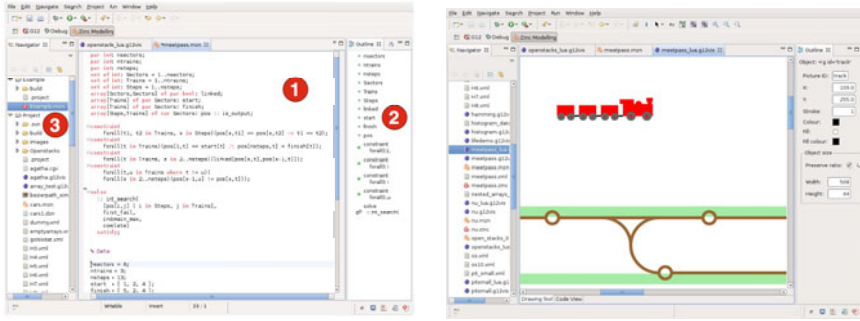
```

Fig. 2. Meet-pass planning problem in MiniZinc

MiniZinc encoding of the problem given that the optimal plan length is known to be 12 moves (13 timesteps). The four constraints are quite simple: the first is a safety condition which could be written using the `all_different` global if we wished; the second is trivial; the third says that trains can't jump and the fourth encodes the remaining safety condition.

4.1 The Zinc Editor

The Zinc Editor (shown in Fig. 3(a)) allows users to write constraint models in the main input languages of the G12 platform: Zinc and MiniZinc [12]. It has a range of features which are typical for code editors in other integrated development environments. For example, automatic syntax checking and highlighting, in-line error reporting, and an outline window to assist with code navigation are standard. Another largely standard feature offered by the Zinc Editor is project-based code management, which simplifies the task of keeping models, data files and related visualisation scripts together. Since the Zinc Editor directly extends the standard Eclipse code editor it is easy to augment its functionality via third party libraries or “plugins”. For example, it is trivial to add support for other programming languages (such as java) or add features to help with revision control.



(a) The Zinc Editor includes features common to many programming environments: syntax highlighting (1), outline views (2) and project-based code management (3) are all standard.

(b) The Visualisation Editor drawing tool. Shown are some domain-specific shapes (trains, tracks) that we have drawn for the Meet-pass problem.

Fig. 3. The two main built-in editors of the G12 IDE

4.2 The Visualisation Editor

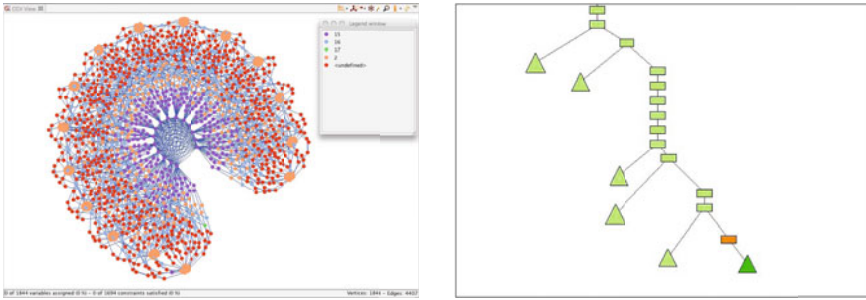
The Visualisation Editor is a vector-oriented drawing tool and an associated script editor which together are used to define arbitrary visualisations of the model as it is being solved.

A new visualisation is created by first drawing a small set of shapes (in the drawing tool) where each shape represents an object or concept specific to the domain of the constraint problem. In the case of Meet-pass for example we might draw a train and some different types of tracks (as shown in Fig. 3(b)). Next, a visualisation script is created which animates the drawn objects – usually in response to some search data received from the solving process. Our visualisation scripts are written in the Lua language and thus both flexible and powerful. We chose Lua because it is portable and lightweight but as we will discuss in Sec. 5 G12IDE can be extended to support any arbitrary programming language.

The basic operation of a visualisation script is straightforward: Each shape defined via the drawing tool is available as a template that can be instantiated by the script. Once an instance is created the shape can be programmatically positioned to anywhere on the canvas. Other attributes of the templated shapes (such as their size, colour, orientation and opacity) can be likewise modified at any point. Further details related to the integration of the scripting language into G12IDE and the operation of the Custom Visualisation Viewer are given in Sec. 5.

5 The Visualisation Layer

G12IDE offers three distinct visualisers: a Constraint Graph Viewer (CGV), a Search Tree Viewer (STV) and a Custom Visualisation Viewer (CVV). The first two are pre-defined and highly optimised visualisations specific to constraint problems and search.



(a) A typical constraint graph. Nodes represent variables and edges, constraints.

(b) A typical search tree. Each node is a domain split. The deltas are collapsed subtrees.

Fig. 4. The search tree and constraint graph visualisers

The third is more general; it can display any arbitrary visualisation which has been developed with the Visualisation Editor (see Sec. 4). We discuss each in turn.

5.1 Pre-defined Visualisations

The STV visualises a constraint solver exploring the search space. It expands and draws a search tree in a step-wise manner that shows decision points, backtracking operations, domains of variables and other information which is useful when inspecting a search tree. Fig. 4(b) shows a typical result. A range of controls is available to speed up, slow down, pause and resume the solving process. It is also possible to collapse and expand entire branches of the tree to speed up drawing time. Additionally, as tree search algorithms can take a long time to complete, we have implemented the “recursive tree estimator” of Kilby et al [10] to provide the user with some indication of how long the solving process is likely to take.

The CGV visualises the solving process using a constraint graph—a useful representation of the structure of a problem. In its simplest form a constraint graph is composed of a set of nodes representing decision variables and a set of edges which represent constraints between them. An alternative representation draws each constraint as a node and adds an edge only if two constraints share a decision variable. CGV supports both types. Updating CGV during the solving process usually amounts to either highlighting or graying out or even hiding instantiated variables and satisfied constraints, and then recovering them on backtracking. This allows the user to see the search progressing through the constraint graph or maybe jumping from one part of it to another. Adding and deleting constraints during the search is less common, but occurs for example when SAT solvers learn nogoods. As the number of constraints and variables in a typical model can be quite large a natural problem which arises is how best to draw the graph. We solve this issue by adopting various force-directed layout algorithms [13,7]. Fig. 4(a) shows a typical result for a scheduling problem. For small graphs, the user may prefer to position nodes by hand: CGV also provides a drag-and-drop facility to support this. The same set of controls used by the STV over the solving process also apply to CGV. In fact, both CGV and STV can be displayed at the same time.

5.2 Custom Visualisations

Both CGV and STV are problem-independent and always available. However, it is often the case that a constraint problem lends itself to a domain-specific visualisation which is more informative. This might show a partial plan for a planning problem, an unfinished Gantt Chart for a scheduling problem or, in the case of one well known CSP problem, queens on a chess board.

Accordingly, we have developed a Custom Visualisation Viewer (CVV) which allows users to create their own domain-specific visualisations in two steps: first, users draw some domain specific objects in the Visual Editor's drawing tool; e.g., a train, a queen, and so forth. Second, users write a short script which controls how these shapes should be drawn on the canvas. To facilitate the creation of such scripts G12IDE offers a very simple API that allows for arbitrary scripting languages to be integrated and used. As a proof of concept we currently provide support for the Lua [8] language.

Any arbitrary visualisation script will, as a minimum, need to implement a `step()` method which will be invoked every time the debugger is paused as a result of reaching some break condition (see Sec. 6 for more details about break conditions). In the course of the `step()` function the script will usually need to call one or both of the following methods (possibly multiple times):

- `g12GetFromData(var)`. This method fetches the current contents of the decision variable `var` and makes its value available to the script.
- `g12Draw(obj, props)`. This method sets a list of properties, `props`, for object `obj`, where `props` may contain items such as the object's positions on the canvas, colour, opacity, scaling factor, etc.

5.3 Support for Other Scripting Languages

The choice to use Lua as the default language of the CVV was a simple one: it is portable, lightweight and often used in similar contexts (for example by the video games industry). Lua also offers an intuitive syntax, straightforward control and data structures as well as a simple type system that can be picked up with minimal effort by anyone familiar with an existing programming language. However, we could as well have chosen Python, Lisp, or any other language which supports Java integration. If besides Lua, we wished to offer, say, Python as a scripting language, we would basically have to implement the above `g12-*` calls in a Python program that in turn invokes the corresponding methods of our IDE. There are only two files specific to the integration of Lua, whose length is less than 1000 loc: the first file is a Lua script which contains the API calls to the IDE, and the second a Java class handling the invocation of the Lua interpreter. How this interaction between the script, the underlying interpreter and the main IDE is technically realised is schematically depicted in Fig. 5. The aforementioned two files are depicted as “Lua Plugin”, written entirely in Java, and “Lua API”, written in Lua. Strictly speaking, the latter is part of the Lua Plugin itself and provided by it. The user who writes a Lua script, merely has to include this file to be able to issue the relevant `g12-*` calls from inside the Lua script.

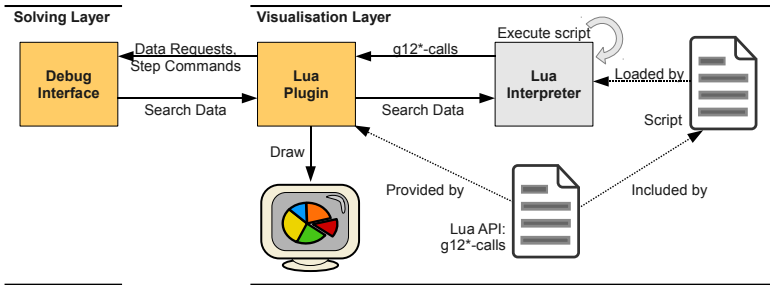


Fig. 5. Schematic overview of visualisation of user-defined scripts. Note that the Lua Plugin provides a bridge between the Custom Visualisation Viewer (not shown) and the Lua Interpreter.

5.4 Performance Debugging with the Custom Visualisation Viewer

Since all visualisations have in common that they are able to be displayed in real-time, i.e., synchronised with relevant events in an ongoing search, they are also often useful for performance debugging. Performance debugging concerns the effectiveness of the encoding of a given problem. For example, in the meet-pass problem, one of the constraints says that a train may not enter a sector if there is already a train there, even if that train is about to move on. This can be read: “you cannot have a track sector occupied by trains at successive times, except in the degenerate case where it’s one train that does not move,” and formulated in Zinc as:

```
constraint
forall(t in 1..ntrains, x in 2..nsteps) (
  pos[x,t] == pos[x-1,t] \ /
  forall(u in 1..ntrains) (pos[x-1,u] != pos[x,t]));
```

While logically correct, this does not cause all the propagation of constraints to happen that one would expect. In particular, at the start of the search, one would expect the solver to have worked out that train 1 (i.e., the left top-most/red train) has to be in track sector 1 at time 2. However, as can be seen in the visualisation, the propagation effectively places trains 1 and 2 on sectors 1 and 2 at time 1, but somehow fails to remove sector 2 from the domain of $\text{pos}[2, 1]$. While it may be possible to deduce this situation merely from looking at the assignments of variables in the debugger’s data view, it is *far* easier to spot in the custom visualisation (see Fig. 6).

Noting this, we rephrased the above requirement as “No two different trains can occupy the same sector at successive times,”:

```
constraint
forall(t,u in Trains where t != u) (
  forall(x in 2..nsteps) (pos[x-1,u] != pos[x,t]));
```

The expected propagation then happens. In our experiments, we have found that custom visualisations like this are not only useful to display results, but also to detect performance bottle-necks that stem from weak propagations like these.



Fig. 6. The missing train: pictures of meet-pass at the start of a search. Propagation should deduce the position of the train at the second timestep (top picture), but in the original formulation (bottom picture) it failed to do so.

6 The Solving Layer

The Solving Layer is comprised of three components: the solver platform itself, a *solver monitor* which collects detailed information about the solving process, and a *debugging interface* which is used to control the stopping and starting of the solving process. Fig. 7 provides an overview of our architecture. The solver monitor and the debugging interface run in separate processes; communication between them is achieved by way of an XML-based messaging system where each message is validated against an external XML Schema². The same schema also defines the set of supported Zinc data types which the monitor recognises and tracks during search.

Conceptually, the communication between solvers and the IDE is purely event-driven. Debugger events are defined separately from the solver code. A number of pre-defined API calls are inserted at the relevant points in the solver to trigger the events if a debugging run has subscribed to them; that is, enabled their sending by the solver when they occur. Events that are subscribed to may simply update the debugger's search model or may also suspend the search until the user decides to continue. At such breakpoints, the front-end debugging tools query the internal model of the search for the information they need by following the XML-based communication protocol between the IDE and

² <http://www.w3.org/XML/Schema>

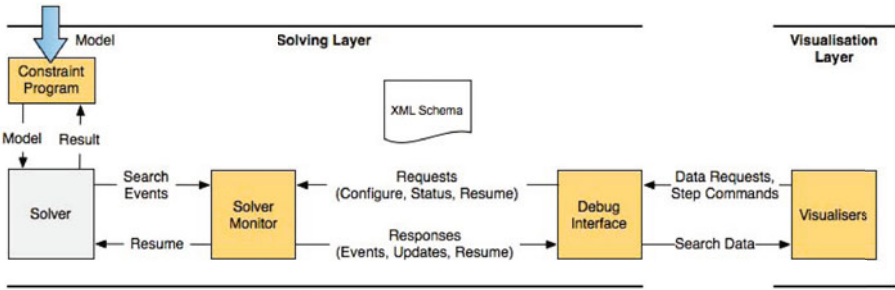


Fig. 7. A closer view of the solving layer. The debugger may be invoked once a model is passed to the constraint program. Note however that the debugger never communicates with the solver directly; rather all communication is via the solver monitor. In addition to controlling the starting and stopping of the solving process the debugger is also responsible for forwarding incremental updates about progress of the solver to the various visualisers.

the solver monitor. The solver adhering to the solver monitor’s API, however, does not have to implement this protocol, which is already handled by the solver monitor.

Our two-process system is highly flexible; by removing any direct dependencies between the solver and interface we are able to transparently substitute one solver for another. A further advantage arising from such a separation is that the solving process can be executed on a completely different (and possibly faster) machine than the one which is running G12IDE. It is entirely possible therefore to commoditize the solving program as an online “service” to which users submit constraint models and from which they receive solutions that can be later visualised in domain-specific terms.

6.1 The Solver Monitor

Invoking the solving process with a debugging flag enables the solver monitor that is attached to the solver. The monitor adheres to a “certainty principle”: it observes the solving process without affecting its course in any way. Before solving commences, a two-way communication channel is opened between the monitor and the debugging interface. We use separate sockets for the requests to and responses from the monitor. This avoids latency problems if a large number of requests need to be sent at one time.

There are three kinds of requests made to the solver monitor: *configuration requests* that modify the state of the monitor by setting and resetting print or break conditions, *status requests* that ask for information about the state of the solver (when stopped) and *resume requests* that cause the solver either to step to the next event or continue until reaching a break condition. Configuration requests may be specific to a particular variable or constraint but can equally apply to an entire class of events. For example, it is possible to pause the search when the solver splits on a particular variable or on any arbitrary variable. Meanwhile a status request can be used to find out the domain of a particular variable or to query the solver about some aspect of its search strategy.

Responses from the monitor include: *solver events*, *status updates* and messages indicating when the solver has stopped. Status updates are generated in direct response

to status requests. Solver events meanwhile are generated whenever the monitor detects that the solving process has reached a point corresponding to a print or break condition. For example, it is possible to generate a solver event message in response to changes to variables and constraints, progression through the search (such as the beginning or end of a propagation phase, or upon reaching a choicepoint or fixpoint) and on the creation of new variables and constraints.

One significant challenge in designing the solver monitor was to resolve the following dilemma: in order to provide maximum flexibility to the visualisation components we need to be able to report detailed information about fine grained search events; at the same time we need to limit the communication between the monitor and the debugging interface as the amount of generated data could easily grow to gigabytes. Our solution is for the solver monitor to maintain an internal table of print and break conditions. Each entry in the table determines what data to send when the solver generates a corresponding search event. Initially the table is empty, optimising the monitor for the “do nothing” case. However, when the solver is paused, configuration requests from the debugging interface are able to modify the table and add (or remove) print or break conditions as required. This way, if a custom visualisation only looks at part of the model it need not incur any overhead associated with processing unrelated events.

6.2 The Debugging Interface

The debugging interface (Fig. 8) provides a variety of mechanisms for controlling the search process and inspecting the model instance under evaluation. A number of “stepping” commands, each operating at varying levels of granularity, are available to progress the search to any given point. Analogues of familiar debugging commands such as “step into”, “step over” and “step return” are available to direct the solver to the next event, the (beginning and end of the) next propagation phase and the next solution respectively.

The familiar Eclipse debugging views, which traditionally show information about the call stack of the current program and the variables on its heap, have been adapted to display information about the solving process. Variables and constraints are shown in an expanding tree, with attributes associated with each such as variable domain or constraint status (whether it is awake, asleep or killed). After each stepping command attributes that have changed are highlighted, similarly to the highlighting of changed data structures in other Eclipse debuggers.

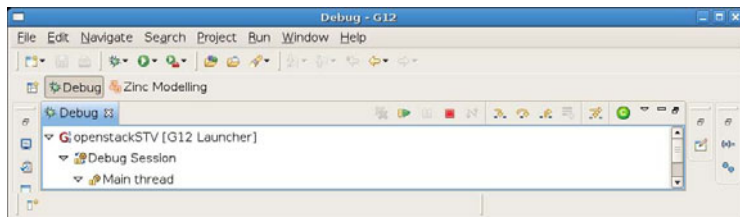


Fig. 8. The debugging controls

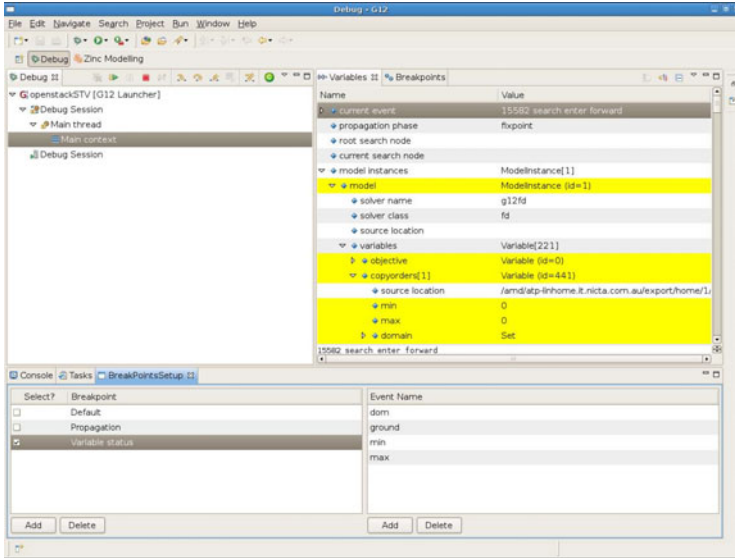


Fig. 9. User-defined breakpoints view. The **C** button advances the solver to the next event specified by the user.

In addition to the pre-defined debugging commands, the IDE allows for custom debugging commands definition, to include only search events that are of interest for the user. For instance, the user might be interested only in changes made to the bounds of decision variables. In such a case, they can create a new *breakpoint* that contains only the “min” and “max” events. The bottom left part of Fig. 9 shows an example of such a breakpoint, called “Variables changed“, whereas the bottom right part shows the search events that the breakpoint is made of. The tick mark next to the breakpoint name shows which breakpoint is currently enabled. Pressing the **C** button sends a request asking for a break every time one of the events belonging to the selected breakpoint occurs.

7 Related Work

There already exist a number of IDEs to support constraint programming systems, some of which provide facilities for visualising constraints in various ways. ECLIPSE CLP [3] for example, allows the modeller to draw a range of charts and graphs of the solver output, whereas ILOG OPL Studio [2] offers a search tree view similar to the one offered by the G12IDE. Choco [1], on the other hand, is more geared towards Java developers who will write their own visualisation tools from scratch, and integrate the Choco Java library as a solver.

There has been work on the use of abstract views of constraints and search to support debugging of constraint programs. The most systematic account is probably that of the DiSCiPl project [4] about a decade ago, which produced a sophisticated search tree viewer and a number of other tools for use with CHIP. OzExplorer [14] was developed at about the same time.

Declarative debugging [11] for Prolog and other logic programming and functional programming languages is related to our approach and could be applied to CP search. It has been studied over many years and many tools were developed. However, it is focused on correctness debugging, which in our experience is rarely the main concern in Zinc modelling. Moreover, the existing tools generally operate at a lower level than those we have developed for G12: we regard debugging a G12 constraint program as a very different activity from debugging the Zinc model.

Drawing tools producing scriptable SVG images are of course well known, and some of them are much more highly developed than the rather simple one we provide. The use of custom views as a way of presenting solutions is also quite standard [9]. However, coupling such a generic tool, including the Lua script editor, to the rest of the debugging package is new with G12IDE.

8 Summary and Conclusions

We presented G12IDE, a front-end for the G12 platform aimed at helping users create and work with constraint models in a manner independent from any underlying solver. Besides offering users an intuitive interface for the G12 platform, it provides advanced features such as debugging a running search and the (custom) visualisation of this process.

Debugging search is in several ways fundamentally different from classical debugging of code. We first had to create a mental model of this process, i.e., define what the relevant concepts and notions are to make this process useful and technically feasible, before attempting an implementation as part of our IDE. Our model is based on a solver monitor which notifies the IDE when relevant events, defined by the user, have occurred in the search to pass control back to the IDE. When this happens, users have the freedom to choose between pre-defined visualisations also known from other, similar IDEs, but also to run their own in this step-wise manner. While supporting this novel model of debugging, we saw it as important to stick as closely as possible to the interface of a classical debugger in order to leverage the application of this new idea. It is also worth noting that there is no opposition between this form of debugging and classical offline profiling: both are useful and one can use both to get a better understanding of how a problem gets solved. Tighter integration of profiling and debugging tools is, in fact, subject to future work and therefore beyond the scope of the present paper.

Custom visualisations, as we have demonstrated using the example of the meet-pass trains, are a useful tool not only to demonstrate high-level and rather coarse properties of a constraint problem, such as the size of its domains, but also to highlight internals of the actual solving process, such as the order of constraint propagation which takes place. As such, visualisations are a useful tool for debugging.

To the best of our knowledge, these core features together with the extensibility of the G12IDE are unmatched in similar front-ends to constraint programming systems. G12IDE is available from the central MiniZinc homepage currently located at the address <http://www.g12.cs.mu.oz.au/minizinc/>.

Acknowledgements. We thank the NICTA G12 team for their valued help and assistance in the development of this work.

References

1. Choco Constraint Solving Toolkit., <http://www.emn.fr/z-info/choco-solver/>
2. ILOG OPL Studio, <http://www.ilog.com/products/oplstudio/>
3. Apt, K., Wallace, M.: Constraint logic programming using ECLiPSe. Cambridge University Press, Cambridge (2007)
4. Deransart, P., Hermenegildo, M.V., Maluszynski, J. (eds.): DiSCiPl 1999. LNCS, vol. 1870. Springer, Heidelberg (2000)
5. Flener, P., Pearson, J., Ågren, M.: Introducing ESRA, a relational language for modelling combinatorial problems. In: Bruynooghe, M. (ed.) LOPSTR 2004. LNCS, vol. 3018, pp. 214–232. Springer, Heidelberg (2004)
6. Gent, I.P., Miguel, I., Rendl, A.: Tailoring solver-independent constraint models: A case study with ESSENCE and MINION. In: Proc. 7th Symposium on Abstraction, Reformulation and Approximation (SARA), pp. 184–199 (2007)
7. Harel, D., Koren, Y.: A fast multi-scale method for drawing large graphs. *J. Graph Algorithms Appl.* 6(3), 179–202 (2002)
8. Ierusalimsky, R., de Figueiredo, L.H., Filho, W.C.: Lua—an extensible extension language. *Softw. Pract. Exper.* 26(6), 635–652 (1996)
9. Jones, C.V.: Visualization and Optimization. Kluwer, Boston (1996)
10. Kilby, P., Slaney, J.K., Thiébaux, S., Walsh, T.: Estimating search tree size. In: Proc. of the Twenty-First National Conference on Artificial Intelligence (AAAI). AAAI Press, Menlo Park (2006)
11. Lee, N.: A declarative debugging scheme. *Journal of Functional and Logic Programming*, 1997(3) (April 1997)
12. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007)
13. Quigley, A.J., Eades, P.: Fade: Graph drawing, clustering, and visual abstraction. In: Graph Drawing, pp. 197–210 (2000)
14. Schulte, C.: Oz explorer: A visual constraint programming tool. In: Proc. of the 14th International Conference on Logic Programming (ICLP), pp. 286–300. MIT Press, Cambridge (1997)
15. Wallace, M., The G12 team: G12 - Towards the Separation of Problem Modelling and Problem Solving. In: van Hoes, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 8–10. Springer, Heidelberg (2009)

Global Constraints on Feature Models

Ahmet Serkan Karataş, Halit Oğuztüzün, and Ali Dođru

Middle East Technical University, Department of Computer Engineering,
06531 Ankara, Turkey
{karatas, oguztuzn, dogru}@ceng.metu.edu.tr

Abstract. Feature modeling has been found very effective for modeling and managing variability in Software Product Lines. The nature of feature models invites, sometimes even requires, the use of global constraints. This paper lays the groundwork for the inclusion of global constraints in automated reasoning on feature models. We present a mapping from extended feature models to constraint logic programming over finite domains, and show that this mapping enables using global constraints on feature attributes, as well as features, for a variety of analysis operations on feature models. We also present performance test results and discuss the benefits of using global constraints.

Keywords: Global constraints, feature models, constraint logic programming over finite domains.

1 Introduction

Production lines aim to reduce the development costs, enhance the quality of products, reduce maintenance efforts and ease project management. Within the recent decades, as software products and software-intensive systems became larger and more complex, Software Product Lines (SPLs) attracted more attention in both academic and industrial communities.

A key concept in SPLs is capturing the *commonality* and the *variability* in the family of the products within the scope. Among different proposals, feature modeling [11] has proven to be very effective for modeling and managing variability in SPLs. A feature is a distinguishable characteristic of a concept (e.g. context, system, component, functionality) that is relevant to some stakeholder of the concept [15]. A Feature Model is a hierarchically arranged set of features, the relationships among these features that determine the composition rules and the cross-tree constraints, and some additional information, such as trade-offs, rationale, and justifications for feature selection.

A number of analysis operations have been proposed to reason about feature models. These operations are used to reveal certain characteristics of a feature model. For instance, questions such as “*how many products can be derived from this feature model?*”, “*which are the core features, which are the variant features?*”, “*is this a valid product with respect to this feature model?*” are answered using analysis operations. However, industrial experiences showed that feature models often grow large with hundreds, even thousands of features and complex cross-tree relationships

among features and attributes of features, which closes the door on manual analysis of feature models. Thus, automated analysis of feature models is a necessary, yet still to be improved issue.

In this paper we explore the use of global constraints in the automated analysis of feature models, which has recently become feasible [13]. We present a mapping from extended feature models to constraint logic programming over finite domains, designated CLP(FD), that enables imposing global constraints on the attributes of features, as well as features. Then, we present some examples, where using some of the well known types of global constraints would significantly simplify the work of both the modeler and the constraint solver, and discuss the benefits of using global constraints. We also present performance results of an off-the-shelf constraint solver on a sample model.

The organization of this paper is as follows. In Section 2, we briefly discuss feature models and automated reasoning on feature models. In Section 3, we state the problem in detail and present the reasons to choose CLP(FD) as the reasoning base. In Section 4, we introduce our mapping proposal, discuss how and when some of the well known global constraints can be used on feature models, and present the performance test results. Section 5 presents conclusions and points to future work.

2 Background



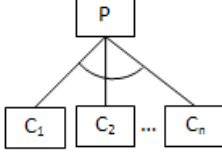
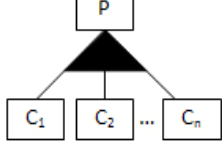
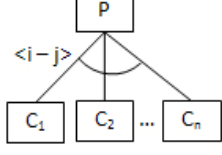

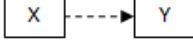
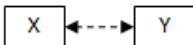
2.1 Feature Models

Feature models have been exceedingly popular in Software Product Line Engineering since their introduction by Kang *et al.* as part of Feature Oriented Domain Analysis (FODA) [11]. After Kang *et al.*'s initial proposal several extensions to feature models have been devised. We refer the reader to [14] for a detailed survey on the feature models. Here we shall briefly discuss basic feature models and their important extensions.

The first type of relationship, which defines the decomposition rules among features, includes four relations: *mandatory relation*, *optional relation*, *alternative relation*, and *or relation*. No child feature in a decomposition relation can be included in a configuration that does not include its parent feature. However, these four relations exhibit different semantics when the parent feature in the relation is included (see Table 1). The second type of relationship, which defines cross-tree constraints in feature models, includes the *requires* and *excludes* relations.

One of the extensions proposed on feature models is the introduction of the UML-style cardinalities, which allows to express multiplicities such as 1, 0..1, 0..*, 1..*, 0..n, 1..n, and so on [14]. These cardinalities are used in decomposition relations. Another important extension has been the introduction of feature attributes to provide further information about features. An attribute of a feature is any characteristic of the feature that can be measured. Every attribute belongs to a domain, the space of possible values where the attribute takes its values [5]. Kang *et al.* mentioned relations between features and feature attributes in [11] and “non-functional” features related to feature attributes in [12]. Using attributes in feature models were introduced

Table 1. Symbols we have used for design of feature models

Symbol	Name	Explanation
	Mandatory Relation	Let P and C be two features, where P is the parent of C, in a <i>mandatory relation</i> . Then, C is included in a configuration if and only if P is included in the configuration.
	Optional Relation	Let P and C be two features, where P is the parent of C, in an <i>optional relation</i> . If P is included in a configuration then C may or may not be included in the configuration.
	Alternative Relation	Let P, C ₁ , C ₂ , ..., and C _n be features, where P is the parent of C ₁ , C ₂ , ..., and C _n , in an <i>alternative relation</i> . If P is included in a configuration, exactly one of the child features C ₁ , C ₂ , ..., C _n must be included in the configuration.
	Or Relation	Let P, C ₁ , C ₂ , ..., and C _n be features, where P is the parent of C ₁ , C ₂ , ..., and C _n , in an <i>or relation</i> . If P is included in a configuration, a nonempty subset of {C ₁ , C ₂ , ..., C _n } must be included in the configuration.
	Group Cardinality	Let P, C ₁ , C ₂ , ..., and C _n be features, where P is the parent of C ₁ , C ₂ , ..., and C _n , in a decomposition with the group cardinality <i - j>. If P is included in a configuration then at least <i>i</i> at most <i>j</i> of the child features must be included in the configuration.
	Feature Attribute (F.attr)	Feature attributes are used to supply some additional information (e.g. priority, cost, size, speed) about features. A feature attribute consists of a name, a domain, and a value (possibly an interval).
	Requires	If a feature X <i>requires</i> a feature Y, the inclusion of X in a configuration implies the inclusion of Y in such a configuration.
	Excludes	If a feature X <i>excludes</i> a feature Y, the inclusion of X in a configuration implies the exclusion of Y in such a configuration.

by Czarnecki *et al.* in [7], where such feature models were called extended feature models. Later Benavides *et al.* [5] proposed a notation for extended feature models. In this paper we have adopted a notation similar to the one proposed by Benavides *et al.* in [5], which is summarized in Table 1.

2.2 Automated Reasoning on Feature Models

Proposals on automated analysis of feature models can be divided into four groups: propositional logic based analyses, description logic based analyses, constraint programming based analyses, and other proposals. We refer the reader to [3] for a detailed literature review on automated analysis of feature models; here we shall briefly discuss the constraint programming based proposals.

Benavides *et al.* [2] were the first to propose using constraint programming to reason on feature models. They have presented an algorithm to transform an extended feature model into a constraint satisfaction problem (CSP) [1] and also provided tool support [4]. In their mapping features make up the set of variables, where the domain of each variable is the set {true, false}, and they represent extra-functional features and relations in the feature model as constraints [2]. The authors have also shown that several kinds of analysis and optimization operations on extended feature models can be supported by means of constraint programming.

However, in some cases feature models contain complex cross-tree constraints including feature attribute relationships such as “*feature 3D Car Race Application requires Memory.size \geq 512*”. We have proposed a mapping from extended feature models to CLP(FD) to handle such cases, and showed that it is possible to take advantage of automated reasoning for the analysis operations on such extended feature models using off-the-shelf CLP(FD) constraint solvers in [13]. We have also presented a sample implementation using the clp(FD) constraint solver [6].

3 The Problem and CP

3.1 Global Restrictions on a Product

During the project management phases it is often the case that developers face requirements that impose restrictions on all or a set of features rather than a feature alone. Requirements such as “*total power consumption of the units in a mobile phone shall not exceed the power supplied by the battery*”, “*total cost of the product must reside in the prescribed interval to suit the targeted budget limits*”, or “*each PCI card to be installed on a computer board must be installed into a different PCI slot for a valid configuration of the computer*” are examples of such restrictions.

Consider the requirement on the installation of the PCI cards into the PCI slots of the computer board, where each PCI card is modeled as a feature and the number of the PCI slot the card is to be installed into is kept in an attribute, *slot no.*, of the PCI card features. This requirement can easily be represented by a clique of inequalities on the *slot no* attributes of the PCI card features. However, for instance if there are 6 PCI cards, it will take 15 binary constraints to represent a single requirement. Also note that sometimes it may not be possible to express some of the global requirements using simple (or any) binary decompositions.

Representing these types of requirements by binary constraints would cause an increase in the number of constraints to represent the feature model, and lead to a tedious coding process for the modeler. Consequently, readability of the code decrease while chances to make errors increase. Moreover, it would also complicate the task of the constraint solver, which would have to deal with a greater number of constraints. As one of the central ideas of constraint programming is the propagation-search technique, the solver would try to detect “dead-ends” as early as possible and run filtering algorithms on each constraint in the CSP [9]. Thus, more constraints would mean necessity for more filtering in most cases.

Clearly using global constraints to represent such requirements on feature models would significantly simplify the task of both the modeler and the constraint solver, as global constraints provide the following advantages [9]: *i)* global constraints provide shorthand for frequently recurring patterns, which simplifies the programming task, *ii)* global constraints facilitate the work of the constraint solver by providing it with a structural view of the problem. Thus, we believe that using global constraints can provide a significant facility for the automated reasoning on feature models.

3.2 Why CP?

As feature models often grow too large to analyze manually, automated reasoning is regarded as the only practical solution. The main reason we have chosen CP as the reasoning base is that, as we discuss in the following section it is possible to map feature attributes to CLP(FD) variables while translating an extended feature model to a CP problem, which enables using global constraints to represent global requirements imposed on the attributes of features, as well as the features.

Another advantage of choosing CP as the reasoning base is the wide variety of tool support [8]. The CLP(FD) solvers have been used for many real-life applications, and evolved to provide efficient implementations for the computationally expensive procedures that had proven to be very effective for modeling discrete optimization and verification problems. The concept of global constraints is a well-studied area in CP, and most of the off-the-shelf solvers provide effective implementation techniques for global constraints. Thus, this enables the application of the techniques presented in this paper by using any CP solver supporting global constraints.

Moreover, due to their declarative style, CLP systems lead to highly readable solutions. CLP systems provide a wide variety of facilities for the users and the global constraints we focus on this paper is one of these. Thus, it becomes straightforward to code many of the important analysis operations (see Section 4.3) in CLP.

4 Global Constraints and Feature Models

4.1 The Mapping

In [13] we have proposed a mapping from extended feature models, which may include complex feature-feature, feature-attribute and attribute-attribute cross-tree relations, to constraint logic programming over finite domains. The two key ideas behind our proposal are *(i)* mapping attributes of the features to CLP variables, and

(ii) representing all the relations in the feature model (i.e. feature decomposition relations and cross-tree relations) as constraints among the features and attributes. In this paper we shall briefly summarize the first idea, which enables us to use global constraints on feature models.

Attributes of features are mapped to CLP(FD) variables using the following rule:

Rule 1 (*Mapping Attributes*): Let F be a feature, a an attribute of F , and $dom F.a$ denote the domain of a , then this attribute is represented with the CLP(FD) variable $F.a \in dom F.a$.

For instance, suppose a computer configuration may include two CPUs with the possible clock speeds (in MHz) $\{1600, 1800, 2000\}$, then the speed attributes of the CPUs can be mapped as follows:

$$CPU1.speed \in \{1600, 1800, 2000\} \wedge CPU2.speed \in \{1600, 1800, 2000\}$$

The attributes may have any domain as long as it is finite. If, for instance some arithmetic constraint must be expressed on the value of an attribute where the domain of the attribute is not compatible (i.e. having nonnumeric values), a conversion may be introduced from the domain of the attribute to a set of integers.

CLP systems provide a large variety of constraints such as arithmetic constraints, membership constraints, propositional constraints, combinatorial constraints, and user-defined constraints. Therefore, once the mapping of the feature attributes to CLP(FD) variables is complete it is possible to represent the relations defined on feature attributes by imposing constraints on these variables. For instance, assume that a requirement states that “*The clock speed of the first CPU cannot be slower than the clock speed of the second CPU*”, then this requirement is represented as follows:

$$CPU1.speed \geq CPU2.speed \tag{1}$$

However, every relation in a feature model involves at least one feature; therefore features must be mapped to CLP(FD) variables as well. In order to treat features and attributes uniformly, we assume that every feature has an implicit attribute named *selected* that ranges over the domain $\{false, true\}$ (or $\{0, 1\}$ respectively). This attribute gets the value *true* if the feature is selected to be included in the product and *false* otherwise. Implicit attributes are defined as follows:

Definition 1 (*Implicit Attribute*): Let \mathcal{FM} be a feature model, and \mathcal{F} denote the set of features in \mathcal{FM} , then each feature $F \in \mathcal{F}$ has an attribute named *selected* with $dom F.selected = \{false, true\}$.

For instance, the implicit attributes of the features *CPU1* and *CPU2* are mapped as:

$$CPU1.selected \in \{false, true\} \wedge CPU2.selected \in \{false, true\}$$

As the value of an attribute is relevant to a constraint only if the feature it belongs to is included in the product, we augment the formula (1) using the implicit attributes as follows:

$$CPU1.speed \geq CPU2.speed \wedge CPU1.selected \wedge CPU2.selected$$

However, as we have discussed in Section 3.1, global requirements are defined on a non-fixed set of feature attributes. As the set of features that will be included in a product may differ from product to product, attributes of those features that have not been selected must be filtered out before checking the satisfaction of the global constraints.

Rule 2 (Filter): Let \mathcal{T} be a set of pairs of the form (b, t) , where b is either *true* or *false*, and t is any term. Then, $filter(\mathcal{T})$ is defined as follows:

$$filter(\mathcal{T}) = \{t \mid (true, t) \in \mathcal{T}\}$$

We utilize the *filter* operation as follows. Let GC be a global constraint defined on a set of attributes \mathcal{A} , and let \mathcal{T} be a set of the pairs $(F.selected, a)$ where $a \in \mathcal{A}$ and F is the feature a belongs to. Our mapping applies $filter(\mathcal{T})$ to find the set of attributes that GC will be applied on by the constraint solver for a particular product.

Note that, any global constraint can alternatively be represented, without making use of the *filter*, by enumerating the possible conditions using the *guarded constraints* (i.e. constraints of the form $Guard \Rightarrow Constraint$) described in [13], as follows:

$$\begin{aligned} & ((F_1.selected \wedge F_2.selected \wedge \dots \wedge F_n.selected) \Rightarrow GC(F_{1.a_1}, \dots, F_{n.a_m})) \wedge \\ & ((\neg F_1.selected \wedge F_2.selected \wedge \dots \wedge F_n.selected) \Rightarrow GC(F_{2.a_1}, \dots, F_{n.a_m})) \wedge \\ & \dots \\ & ((F_1.selected \wedge \neg F_2.selected \wedge \dots \wedge \neg F_n.selected) \Rightarrow GC(F_{1.a_1}, \dots, F_{1.a_k})) \end{aligned}$$

Definition 2 (Neutral Value): Let GC be a global constraint defined on a set of variables $X = \{x_1, \dots, x_n\}$, and \mathcal{V} a subset of X . Let \mathcal{B} be a set of bindings of the form v_i / x_i , where each x_i in \mathcal{V} occurs exactly once and v_i is a ground term. \mathcal{B} is said to be *neutralizing* \mathcal{V} for GC if the set of solutions for GC applied on X with \mathcal{B} is the same as the set of solutions for GC applied on $X - \mathcal{V}$. In this situation each v_i is called a *neutral value* for GC .

Global constraints often possess neutral values. For instance, for the global constraints *sum* and *knapsack* (see the following section) 0 acts as a neutral value, which happens to be the only one. Assume that an attribute is involved in a *sum* constraint, but the feature it belongs to is not selected. Clearly this attribute must not figure in checking the satisfaction of the constraint. However, if the attribute has the value 0 , the additive identity, the result would not be affected with the attribute figured in. Hence, as we discuss in the remainder of this section, under certain circumstances it would be possible to get the same effect without applying the *filter*, thereby avoiding the overhead incurred by filtering.

Definition 3 (Common Neutral Value): Let $F.a$ be an attribute involved in one or more global constraints. A *Common Neutral Value* for the attribute $F.a$ is a neutral value with respect to each of these global constraints. Note that, a common neutral value, designated $v_{F.a}$, may or may not be an element of $dom F.a$.

Note that a common neutral value is defined only for the attributes involved in one or more global constraints. As it is the case for the illustrative example presented in Section 4.3, it is often possible to find common neutral values for the attributes involved in global constraints. However, if it is not possible to find a common neutral

value for some of the attributes (e.g. for an attribute that is involved in two global constraints where the value of the attribute is involved in a summation in one of the constraints and in a multiplication in the other), one can simply make use of the *filter* on such attributes, which is the general solution for applying global constraints.

Next, we revise the mapping rule for attributes with a common neutral value.

Rule 3 (*Mapping an Attribute with a Common Neutral Value*): Let F be a feature, a an attribute of F with the domain $dom F.a$ and the common neutral value $V_{F.a}$. Then, the attribute is mapped as follows:

$$F.a \in dom F.a \cup \{V_{F.a}\} \wedge \\ (F.selected \Rightarrow F.a \in dom F.a) \wedge (\neg F.selected \Rightarrow F.a = V_{F.a})$$

4.2 Using Global Constraints

In this section we discuss how and when some of the well-known types of global constraints provided by the CLP systems can be used in the automated reasoning on feature models. Note that the examples presented in the following subsections do not provide an exhaustive listing of the global constraints that can be used on feature models. As the nature of feature models invites, sometimes even requires, the use of global constraints, we impose no limitations on the number or types of global constraints to be used. Once the feature model is translated to a CP program using the mapping discussed in the previous section, the modeler can use any global constraint provided by the solver, and if the solver provides such a facility, may also define his/her own global constraints.

4.2.1 The Sum Constraint

The *sum* constraint is one of the most common constraints used in applications. The definition of the sum constraint is as follows [9]:

$$sum(x_1, \dots, x_n, z, c) \equiv z = \sum_{i=1}^n c_i x_i$$

As an example, consider the sample feature diagram, which is a part of a feature model for a mobile phone, given in Figure 1. Assume that a global requirement, GR , states that; *total power consumption of the electronic components in the mobile phone cannot exceed the power supplied by the battery*. This global requirement can be represented using the *sum* constraint as follows:

$$FS = filter((Bluetooth.selected, Bluetooth.pc), \dots, (Camera.selected, Camera.pc)) \\ \wedge sum(FS, total, 1) \wedge (total \leq Battery.power)$$

An alternative representation, if the attributes involved in GR has 0 as the *common neutral value*, would be:

$$sum(Bluetooth.pc, CPU.pc, GPS.pc, RAM.pc, Screen.pc, Camera.pc, total, 1) \\ \wedge (total \leq Battery.power)$$

The attributes involved in the *sum* constraint are, then, mapped as:

$$(Bluetooth.selected \Rightarrow Bluetooth.pc \in \{\dots\}) \wedge (\neg Bluetooth.selected \Rightarrow Bluetooth.pc = 0) \\ \wedge Bluetooth.pc \in \{\dots\} \cup \{0\} \wedge \dots \wedge Camera.pc \in \{\dots\} \cup \{0\} \wedge \\ (Camera.selected \Rightarrow Camera.pc \in \{\dots\}) \wedge (\neg Camera.selected \Rightarrow Camera.pc = 0)$$

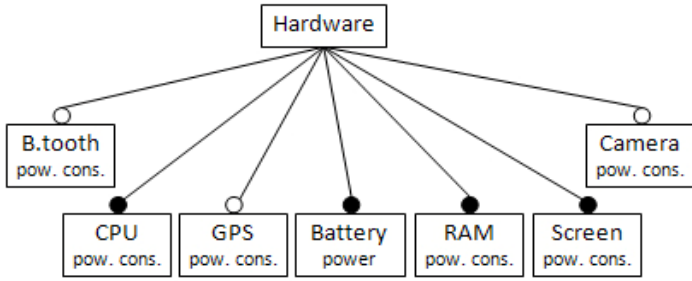


Fig. 1. Part of a sample feature model for a mobile phone

4.2.2 The Knapsack Constraint

The *knapsack* constraint is a variant of the *sum* constraint. It requires the sum be within a lower bound l and upper bound u . It is defined as follows [9]:

$$\text{knapsack}(x_1, \dots, x_n, z, c) \equiv \min D(z) \leq z = \sum_{i=1}^n c_i x_i \leq \max D(z)$$

As an example, consider the sample feature diagram, which is a part of a feature model for a mobile phone, given in Figure 2. Assume that a global requirement states that; *for the mobile phone products targeting the medium budget segment, total cost of the electronic components must be greater than or equal to 200USD and less than or equal to 400 USD*. This requirement can be represented using the *knapsack* constraint as follows:

$$\text{FS} = \text{filter}((\text{Btooth.selected}, \text{Btooth.cost}), \dots, (\text{Camera.selected}, \text{Camera.cost})) \\ \wedge \text{total} \in [200, 400] \wedge \text{knapsack}(\text{FS}, \text{total}, 1)$$

Note that 0 would be the *common neutral value* for the attributes involved in a *knapsack* constraint, if it is possible (depending on other global constraints on these attributes) to introduce a *common neutral value* for such attributes.

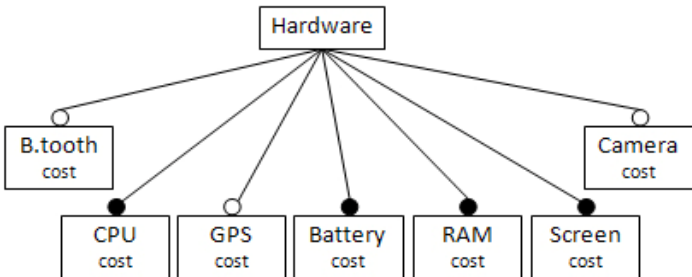


Fig. 2. Part of a sample feature model for a mobile phone

4.2.3 The Alldifferent Constraint

The *alldifferent* constraint is one of the best-known, most influential and most studied global constraints in the constraint programming, and it is defined as follows [9]:

$$alldifferent(x_1, \dots, x_n) = \{(d_1, \dots, d_n) | \forall i d_i \in D(x_i), \forall i \neq j d_i \neq d_j\}$$

where x_1, \dots, x_n are variables.

As an example, consider the sample feature diagram, which is a part of a feature model for a computer family, given in Figure 3.

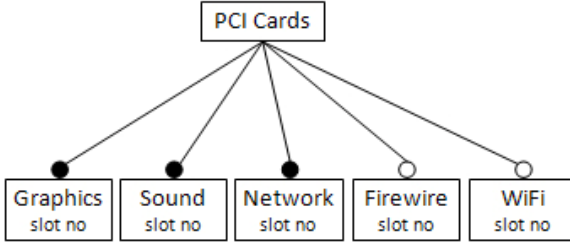


Fig. 3. Part of a sample feature model for a computer

Assume that a number of PCI Cards may be included in a computer. Clearly each PCI Card must be installed into a different PCI Slot. This requirement can be easily represented using the *alldifferent* constraint as follows:

$$FS = filter((Graphics.selected, Graphics.sn), \dots, (WiFi.selected, WiFi.sn)) \wedge alldifferent(FS)$$

Alternatively n distinct *common neutral values* such that;

$$v_i \notin \text{dom } x_1 \cup \dots \cup \text{dom } x_{i-1} \cup \text{dom } x_{i+1} \cup \dots \cup \text{dom } x_n$$

can be used for the attributes involved in an *alldifferent* constraint, if it is possible (depending on other global constraints on these attributes) to introduce such *common neutral values*.

4.2.4 The Global Cardinality Constraint

The *global cardinality constraint* $gcc(x_1, \dots, x_n, c_{v_1}, \dots, c_{v_n'})$ is a generalization of the *alldifferent* constraint. There are n' count variables, c_{v_1} through $c_{v_n'}$, and GCC requires that each value v_i is assigned to exactly c_{v_i} assignment variables, where x_1 through x_n make the set of assignment variables. The *global cardinality constraint* is defined as follows [9]:

$$gcc(x_1, \dots, x_n, c_{v_1}, \dots, c_{v_n'}) = \{(w_1, \dots, w_n, o_{v_1}, \dots, o_{v_n'}) | \forall j w_j \in D(x_j), \forall i occ(v_i, (w_1, \dots, w_n)) = o_i \in D(c_{v_i})\}$$

Where, x_1, \dots, x_n are assignment variables whose domains are contained in $\{v_1, \dots, v_n'\}$, $\{c_{v_1}, \dots, c_{v_n'}\}$ are count variables and $occ(v, t)$ is the number of occurrences of v in t .

As an example, consider the sample feature diagram, which is a part of a feature model for a software intensive family, given in Figure 4.

In the sample case a product includes 3 communication channels to be used by the n communication routines. Each channel has a designated capacity, say $c_1, c_2,$ and c_3

that should not be exceeded. A valid product must obey these constraints. These requirements can be represented using the *global cardinality constraint* as follows:

$$\begin{aligned} \text{FS1} &= \text{filter}((\text{CR 1.selected}, \text{CR 1.ch}), \dots, (\text{CRN.selected}, \text{CR N.ch})) \wedge \\ \text{FS2} &= \text{filter}((\text{Chan1.selected}, (1, v_1)), \dots, (\text{Chan3.selected}, (3, v_3))) \wedge \\ &\quad \text{gcc}(\text{FS1}, \text{FS2}) \wedge v_1 \in [0, c_1] \wedge v_2 \in [0, c_2] \wedge v_3 \in [0, c_3] \end{aligned}$$

Note that 0 would be the *common neutral value* for v_1 , v_2 , and v_3 if it is possible.

Similarly, it is straightforward to apply the *global cardinality constraint with costs*, when there is a cost function in the model.

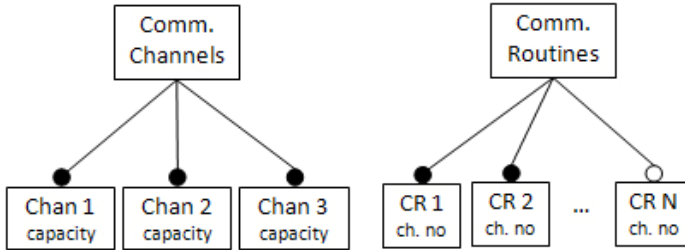


Fig. 4. Part of a sample feature model for a software intensive family

4.3 A Sample Case

As an illustrative example consider the sample feature model, designed for a computer product family, given in Figure 5. The feature model has one *concept feature* (the root feature), 28 *mandatory features*, 12 *optional features*, 8 *alternative features*, and 3 *or features*. Assume we have the following five global constraints:

- Each PCI Card that is a part of the product must be installed on a different PCI slot.
- Each data collector, 1 through 4, must be assigned to a data communication channel, 1 through 3. Each data communication channel has a designated capacity and these capacities must not be exceeded.
- Total power consumption of the hardware parts cannot exceed the capacity of the power supply.
- Total cost of a product cannot exceed a designated budget.
- Memory size must be greater than the total memory consumption of the system software plus memory consumption of the application with the highest memory requirement among the applications chosen to be a part of the product.

In addition to these global constraints we have some cross-tree constraints such as:

- *Task Scheduler* requires $\text{CPU 1.speed} \geq \text{CPU 2.speed}$.
- *Data Collectors 1* and *2* cannot be assigned to the same channel.

We have implemented the following analysis operations on the sample feature model \mathcal{FM} . These operations are defined in [3], on an arbitrary feature model, say \mathcal{M} :

- **Void Feature Model:** Does \mathcal{M} represent any products?
- **Valid Product:** Does the given product belong to the set of products represented by \mathcal{M} ?

- **Valid Partial Configuration:** Is the given partial configuration valid (i.e. does not include any contradiction) with respect to \mathcal{M} ?
- **All Products:** Compute all products represented by \mathcal{M} .
- **Number of Products:** Compute the number of products represented by \mathcal{M} .
- **Commonality:** Compute the percentage of products represented by \mathcal{M} including a given configuration.
- **Filter:** Compute all products, including a given configuration, represented by \mathcal{M} .
- **Core Features:** Compute the set of features that are part of all the products.
- **Variant Features:** Compute the set of features that appear in some but not all of the products.
- **Dead Features:** Compute the set of features that are not part of any product.
- **False Optional Features:** Compute the set of features that, although modeled as optional, are part of all the products.
- **Optimization:** Compute the products fulfilling the criteria established by the given objective function.

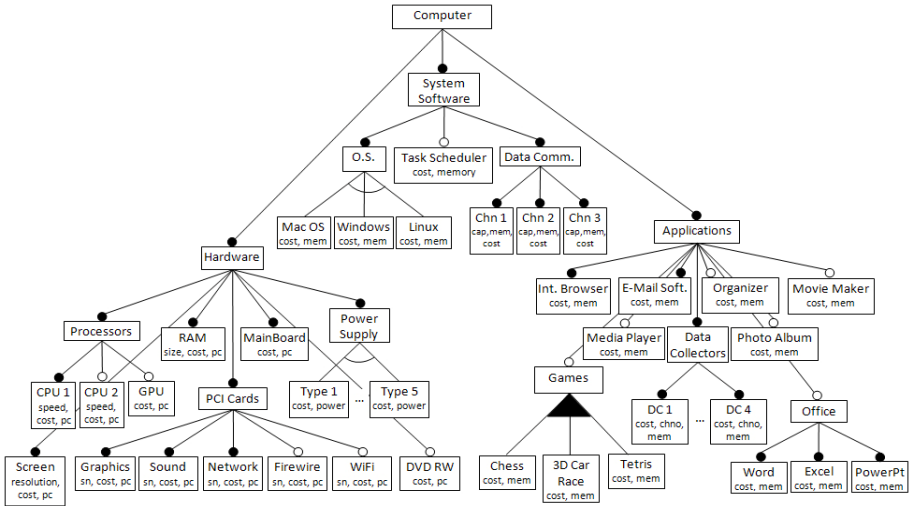


Fig. 5. A sample feature model, \mathcal{FM} , for a computer product family

For implementation we have used `clp(FD)` [6], which is available as a library module for SICStus Prolog [10]. The `clp(FD)` provides a large variety of global constraint library predicates and we have used some of them such as *all_different*, *global_cardinality*, *sum*, *maximum*, *labeling* and so on to represent the global constraints on the feature model. The tool also provides a facility to introduce user-defined global constraints but we have not used this option as the library global constraints were sufficient for the present modeling and analysis tasks. We have also expressed the global constraints without making use of the library predicates in a separate equivalent program to observe their effect on the performance (see Table 2).

Table 2. Performance results for the analysis operations

Analyses Operation	≈ Time (seconds)		Gain
	With library GC predicates	Without library GC predicates	
Void feature model	0.015	0.023	35%
Valid product	0.005	0.005	-
Valid partial configuration	0.015	0.022	32%
All products	16.305	20.360	20%
Number of products	16.322	20.375	20%
Commonality ¹	26.609	33.427	21%
Filter ¹	10.422	12.974	20%
Core features	0.217	0.362	40%
Variant features	0.218	0.364	40%
Dead features	0.218	0.363	40%
False optional features	0.218	0.364	40%
Optimization (<i>maximize cost</i>)	0.109	0.246	56%
Optimization (<i>minimize cost</i>)	0.089	0.234	62%

We have performed the tests on a computer with an Intel Core 2 Duo T5500 1.66 GHz CPU and 2 GB RAM, and running Microsoft Windows XP Professional. Note, however, that although the computer had 2 GB of physical memory, the SICStus Prolog version we have could utilize only 256 MB of memory on 32 bit systems [10].

The tool automatically derived 338,928 products from the sample feature model. For the operations *number of products* and *commonality* we have generated the set of all products to find the answer. It should be possible to implement these operations without actually generating the set of all products; however this is not an issue we are intending to address in the present work. For the optimization operation we have asked the tool to derive *the most* and *the least expensive* products. Performance results for the aforementioned analysis operations are presented in Table 2.

5 Discussion and Conclusion

Our experiments suggest that using global constraints can simplify the task of model validation, for both the modeler and the constraint solver, by providing an elegant way to express global project requirements. For instance, we had to introduce 16 constraints, each being conjunctions of 15 binary constraints and an implication, to decompose the global constraint *maximum* that was defined on 16 feature attributes, which increased the number of constraints to be considered by the solver while decreasing the readability and maintainability of the code.

¹ Time for these operations heavily depend on the input configuration. For both of the operations we have used “*products including CPU 2*” as the input configuration.

Comparative performance test results presented in Table 2 show that using the global constraint predicates has paid-off in terms of performance measures as well. As the results indicate, using global constraints increased efficiency up to 20% in the most time consuming analysis operations such as *all products*, *commonality*, and *filter*. The increase in the efficiency was even above 50% in optimization operations.

In this paper we have focused on application of the CP technology to the field of software product line engineering. As the mapping from extended feature models to CLP(FD) has a well-defined structure, it is possible to incorporate it into existing feature modeling and analysis tools or develop new tools. Such a tool should comprise the following components:

- A *file i/o component* to read in the feature models represented in an input language. The input language must be rich enough to represent the features, attributes of the features and their domains, decomposition relations among features, cross-tree relations among features and attributes of features, and widely used global constraints. Such a language can be defined as an extension of a feature markup language [4, 5].
- A *preprocessor* to carry out such tasks as finding possible common neutral values for attributes involved in global constraints, possibly with user guidance, and introducing the implicit attributes.
- A *mapping component* to perform the translation to targeted CLP(FD) notation.
- An *analysis component* that will utilize an off-the shelf constraint solver for the analysis operations.
- A *postprocessor* to output and comment on the results produced by the solver.

As product line engineers may not necessarily be competent in constraint solving, the tool must employ a user-friendly interface, and hide the details of the solver as much as possible while enabling the users to communicate their constraints to the tool. It seems worthwhile to design a language (or extend existing ones) to represent extended feature models with complex cross-tree and global constraints and implement the envisioned tool.

References

1. Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Coping with automatic reasoning on software product lines. In: Proceedings of the 2nd Groningen Workshop on Software Variability Management (November 2004)
2. Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Using constraint programming to reason on feature models. In: The Seventeenth International Conference on Software Engineering and Knowledge Engineering, SEKE 2005, pp. 677–682 (2005)
3. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated Analysis of Feature Models: A Detailed Literature Review. *Information Systems* 35(6), 615–636 (2010)
4. Benavides, D., Segura, S., Trinidad, P., Ruiz-Cortés, A.: FAMA: Tooling a framework for the automated analysis of feature models. In: Proceeding of the First International Workshop on Variability Modeling of Software-intensive Systems (VAMOS), pp. 129–134 (2007)

5. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated Reasoning on Feature Models. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 491–503. Springer, Heidelberg (2005)
6. Carlsson, M., Ottosson, G., Carlson, B.: An Open-Ended Finite Domain Constraint Solver. In: Proc. Programming Languages: Implementations, Logics, and Programs (1997)
7. Czarnecki, K., Bednasch, T., Unger, P., Eisenecker, U.: Generative programming for embedded software: An industrial experience report. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, pp. 156–172. Springer, Heidelberg (2002)
8. Fernandez, A., Hill, P.M.: A comparative study of eight constraint programming languages over the Boolean and finite domains. *Journal of Constraints* 5, 275–301 (2000)
9. van Hoes, W., Katriel, I.: Global Constraints. In: *Handbook of Constraint Programming*. Elsevier, Amsterdam (2006)
10. <http://www.sics.se/isl/sicstuswww/site/index.html>
11. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analyses (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh (1990)
12. Kang, K., Kim, S., Lee, J., Kim, K.: FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering* 5, 143–168 (1998)
13. Karataş, A.S., Oğuztüzin, H., Doğru, A.: Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains. To be Presented in the 14th International Software Product Line Conference (SPLC 2010), South Korea (2010)
14. Schobbens, P., Trigaux, J.C., Heymans, P., Bontemps, Y.: Generic semantics of feature diagrams. *Computer Networks* 51(2), 456–479 (2007)
15. Simos, M., et al.: Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook Version 2.0. In: STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, Manassas, VA (1996)

Constraint Programming for Mining n-ary Patterns

Mehdi Khiari, Patrice Boizumault, and Bruno Crémilleux

GREYC, CNRS - UMR 6072, Université de Caen Basse-Normandie,
Campus Côte de Nacre, F-14032 Caen Cedex, France
{Forename.Surname}@info.unicaen.fr

Abstract. The aim of this paper is to model and mine patterns combining several local patterns (n-ary patterns). First, the user expresses his/her query under constraints involving n-ary patterns. Second, a constraint solver generates the correct and complete set of solutions. This approach enables to model in a flexible way sets of constraints combining several local patterns and it leads to discover patterns of higher level. Experiments show the feasibility and the interest of our approach.

1 Introduction

Knowledge Discovery in Databases involves different challenges, such as the discovery of patterns of a potential user's interest. The constraint paradigm brings useful techniques to express such an interest. If mining local patterns under constraints is now a rather well-mastered domain including generic approaches [3], these methods do not take into account the interest of a pattern with respect to the other patterns which are mined: the useful patterns are lost among too much trivial, noisy and redundant information. In practice, a lot of patterns which are expected by the data analyst (cf. Section 2.2) require to consider simultaneously several patterns to combine the fragmented information conveyed by the local patterns. It also explains why the question of how to turn collections of local patterns into global models such as classifiers or clustering receives a large attention [13]. That is why the discovery of patterns under constraints involving combinations of local patterns is a major issue. In the following, such constraints are called *n-ary constraints* and they define n-ary patterns.

There are very few attempts on mining patterns involving several local patterns and the existing methods tackle particular cases by using devoted techniques [19]. One explanation of the lack of generic methods may be the difficulty of the task: mining local patterns under constraints requires the exploration of a large search space but mining patterns under n-ary constraints is even harder because we have to take into account and compare the solutions satisfying each pattern involved in the constraint. We think that the lack of generic approaches restrains the discovery of useful patterns because the user has to develop a new method each time he wants to extract a new kind of patterns. It explains why this issue deserves our attention.

In this paper, we propose a generic approach for modeling and mining n-ary patterns using Constraint Programming (CP). Our approach proceeds in two steps. First, the user specifies the set of constraints which has to be satisfied. Such constraints handle set operations (e.g., inclusion, membership) but also numeric properties such as the frequency or the length of patterns. A constraint solver generates the correct and complete set of solutions. The great advantage of this modeling is its flexibility, it enables us to define a large broad of n-ary queries leading to discover patterns of higher level. It is no longer necessary to develop algorithms from scratch to mine new types of patterns. To the best of our knowledge, it is the first generic approach to express and mine patterns involving several local patterns.

Cross-fertilization between data mining and CP is a research field in emergence and there are very few attempts in this area. A seminal work [8] proposes a formulation in CP of constraints on local patterns but it does not address rich patterns such as n-ary patterns. We did a preliminary work [12] based on a joint use of local patterns constraint-based mining and CP in order to discover n-ary patterns and to investigate such relationships.

This paper is organized as follows. Section 2 sketches definitions and presents the context. Our approach is described in Section 3, we illustrate it from several kinds of n-ary queries. Section 4 describes the modeling of such n-ary queries as Constraint Satisfaction Problems (CSP). Section 5 presents a background on pattern discovery and our hybrid method for mining n-ary patterns. In Section 6, we go further in CP area and we propose a *full-CP* method to mine n-ary queries. Section 7 compares the two methods. Finally we conclude (Section 8) giving several research issues on using CP for pattern discovery.

2 Definitions and First Examples

2.1 Local Patterns

Let \mathcal{I} be a set of distinct literals called items, an itemset (or *pattern*) is a non-null subset of \mathcal{I} . The language of itemsets corresponds to $\mathcal{L}_{\mathcal{I}} = 2^{\mathcal{I}} \setminus \emptyset$. A *transactional dataset* is a multi-set of itemsets of $\mathcal{L}_{\mathcal{I}}$. Each itemset, usually called a *transaction* or object, is a database entry. For instance, Table 1 gives a transactional dataset \mathbf{r} where 9 objects o_1, \dots, o_9 are described by 6 items A, \dots, c_2 .

Constraint-based mining task selects all the itemsets of $\mathcal{L}_{\mathcal{I}}$ present in \mathbf{r} and satisfying a predicate which is named *constraint*. *Local patterns* are regularities that hold for a particular part of the data. Here, locality refers to the fact that checking whether a pattern satisfies or not a constraint can be performed independently of the other patterns holding in the data.

Example. Let X be a local pattern. The *frequency* constraint focuses on patterns occurring in the database a number of times exceeding a given minimal threshold: $freq(X) \geq minfr$. An other interesting measure to evaluate the relevance of patterns is the *area* [15]. The *area* of a pattern is the product of its frequency times its length: $area(X) = freq(X) \times length(X)$ where $length(X)$ denotes the cardinality of X .

Table 1. Example of a transactional context r

Trans.	Items
o_1	$A \ B \ c_1$
o_2	$A \ B \ c_1$
o_3	$C \ c_1$
o_4	$C \ c_1$
o_5	$C \ c_1$
o_6	$A \ B \ C \ D \ c_2$
o_7	$C \ D \ c_2$
o_8	$C \ c_2$
o_9	$D \ c_2$

2.2 N-ary Patterns

In practice, the data analyst is often interested in discovering richer patterns than local patterns and he/she is looking for patterns that reveal more complex characteristics from the database. The definitions relevant to such patterns rely on properties involving several local patterns and are formalized by the notions of *n-ary constraint* and *n-ary pattern* leading to *n-ary queries*:

Definition 1 (n-ary constraint). *A constraint q is said n-ary if it involves several local patterns.*

Definition 2 (n-ary pattern). *A pattern X is said n-ary if it is defined by a n-ary constraint.*

Definition 3 (n-ary query). *A n-ary query is a conjunction of n-ary constraints.*

2.3 Motivating Examples

N-ary queries enable us to design a lot of patterns requested by the users such as the discovery of pairs of exception rules without domain-specific information [19] or the simplest rules in the classification task based on associations [21].

Example 1. An exception rule is defined as a pattern combining a strong rule and a deviational pattern to the strong rule, the interest of a rule of the pattern is highlighted by the comparison with the other rule. The comparison between rules means that these exception rules are *not* local patterns. This enables us to distinguish exception rules from rare rules where a rare rule is a rule with a very low frequency value. This is useful because in practice rare rules cannot be straightforwardly used because many of them arise by chance and are unreliable. More formally, an exception rule is defined within the context of a pair of rules as follows (I is an item, for instance a class value, X and Y are local patterns):

$$e(X \rightarrow \neg I) \equiv \begin{cases} true & \text{if } \exists Y \in \mathcal{L}_{\mathcal{I}} \text{ such that } Y \subset X, \text{ one have } (X \setminus Y \rightarrow I) \wedge (X \rightarrow \neg I) \\ false & \text{otherwise} \end{cases}$$

Such a pair of rules is composed of a common sense rule $X \setminus Y \rightarrow I$ and an exception rule $X \rightarrow \neg I$ since usually if $X \setminus Y$ then I . The exception rule isolates unexpected information. This definition assumes that the common sense rule has a high frequency and a rather high confidence and the exception rule has a low frequency and a very high confidence (the confidence of a rule $X \rightarrow Y$ is $freq(X \cup Y) / freq(X)$). Assuming that a rule $X \rightarrow Y$ holds iff at least 2/3 of the transactions containing X also contains Y , the rule $AC \rightarrow \neg c_1$ is an exception rule in our running example (see Table 1) because we jointly have $A \rightarrow c_1$ and $AC \rightarrow \neg c_1$. Note that Suzuki proposes a method based on sound pruning and probabilistic estimation [19] to extract the exception rules, but this method is devoted to this kind of patterns.

Example 2. In the context of genomics, data are often noisy and the search of fault-tolerant patterns is very useful to cope with the intrinsic uncertainty embedded in the data [2]. Defining n-ary queries is a way to design such fault-tolerant patterns candidate to be synexpression groups: larger sets of genes with few exceptions are expressed by the union of several local patterns satisfying the *area* constraint previously introduced and having a large overlapping between them. It corresponds to the following n-ary query:

$$(area(X) > min_{area}) \wedge (area(Y) > min_{area}) \wedge (area(X \cap Y) > \alpha \times min_{area})$$

where min_{area} denotes the minimal area and α is a threshold given by the user to fix the minimal overlapping between the local patterns X and Y .

3 Examples of n-ary Queries

In this section, we present the modeling of several n-ary queries within our approach. Some of them were introduced in Section 2.2.

3.1 Exception Rules

Let $freq(X)$ be the frequency value of the pattern X . Let Y be a pattern, I and $\neg I \in \mathcal{I}$ (I and $\neg I$ can represent two class values of the dataset). Let $minfr$, $maxfr$, $\delta_1, \delta_2 \in \mathbb{N}$. The exception rule n-ary query is formulated as it follows:

- $X \setminus Y \rightarrow I$ is expressed by the conjunction: $freq((X \setminus Y) \sqcup I) \geq minfr \wedge (freq(X \setminus Y) - freq((X \setminus Y) \sqcup I)) \leq \delta_1$ which means that $X \setminus Y \rightarrow I$ must be a frequent rule having a high confidence value.
- $X \rightarrow \neg I$ is expressed by the conjunction: $freq(X \sqcup \neg I) \leq maxfr \wedge (freq(X) - freq(X \sqcup \neg I)) \leq \delta_2$ which means that $X \rightarrow \neg I$ must be a rare rule having a high confidence value.

¹ The symbol \sqcup denotes the disjoint union operator. It states that for a rule, patterns representing respectively premises and conclusion must be disjoint.

To sum up:

$$exception(X, Y, I) \equiv \begin{cases} \exists Y \subset X \text{ such that:} \\ freq((X \setminus Y) \sqcup I) \geq minfr \wedge \\ (freq(X \setminus Y) - freq((X \setminus Y) \sqcup I)) \leq \delta_1 \wedge \\ freq(X \sqcup \neg I) \leq maxfr \wedge \\ (freq(X) - freq(X \sqcup \neg I)) \leq \delta_2 \end{cases}$$

3.2 Unexpected Rules

Padmanabhan and Tuzhilin [17] propose the notion of *unexpected* rule $X \rightarrow Y$ with respect to a belief $U \rightarrow V$ where U and V are patterns. Basically, an unexpected rule means that Y and V logically contradict each other. It is defined in [17] by (1) $Y \wedge V \models False$, (2) $X \wedge U$ holds (it means XU frequent), (3) $XU \rightarrow Y$ holds ($XU \rightarrow Y$ frequent and has a sufficient confidence value), (4) $XU \rightarrow V$ does not hold ($XU \rightarrow V$ not frequent or $XU \rightarrow V$ has a low confidence value). Given a belief $U \rightarrow V$, an unexpected rule $un.(X, Y)$ is modeled by the following n-ary query:

$$un.(X, Y) \equiv \begin{cases} freq(Y \cup V) = 0 \wedge \\ freq(X \cup U) \geq minfr_1 \wedge \\ freq(X \cup U \cup Y) \geq minfr_2 \wedge (freq(X \cup U \cup Y) / freq(X \cup U)) \geq minconf \wedge \\ (freq(X \cup U \cup V) < maxfr \vee (freq(X \cup U \cup V) / freq(X \cup U)) < maxconf) \end{cases}$$

3.3 Synexpression Groups

From n local patterns, the search of synexpression groups is expressed by the following n-ary query:

$$synexpr(X_1, \dots, X_n) \equiv \begin{cases} \forall 1 \leq i < j \leq n, \\ area(X_i) > min_{area} \wedge \\ area(X_j) > min_{area} \wedge \\ area(X_i \cap X_j) > \alpha \times min_{area} \end{cases}$$

where min_{area} denotes the minimal area (defined in Section 2.1) and α is a threshold given by the user to fix the minimal overlapping between the local patterns. This example illustrates how a n-ary query enables us to easily express complex and fault-tolerant patterns such as candidate synexpression groups.

3.4 Classification Conflicts

In classification based on associations [21], the quality of the classifier is based on the combination of its local patterns. By taking into account several local patterns, n-ary queries are very useful to help to design classifiers. Let c_1 and c_2 be the items denoting the class values. A classification conflict can be defined by a pair of frequent rules $X \rightarrow c_1$ and $Y \rightarrow c_2$ with a confidence greater than a minimal threshold $minconf$ and a large overlapping between their premises.

$$classif. conflict(X, Y) \equiv \begin{cases} freq(X) \geq minfr \wedge \\ freq(Y) \geq minfr \wedge \\ (freq(X \sqcup \{c_1\}) / freq(X)) \geq minconf \wedge \\ (freq(Y \sqcup \{c_2\}) / freq(Y)) \geq minconf \wedge \\ 2 \times length(X \cap Y) \geq (length(X) + length(Y)) / 2 \end{cases}$$

Table 2. Primitive constraints for the exception rules

N-ary constraints	Primitive constraints
$freq((X \setminus Y) \sqcup I) \geq minfr$	$F_2 \geq minfr$
	$\wedge I \in X_2$
$freq(X \setminus Y) - freq((X \setminus Y) \sqcup I) \leq \delta_1$	$\wedge X_1 \subsetneq X_3$
	$F_1 - F_2 \leq \delta_1$
$freq(X \sqcup \neg I) \leq maxfr$	$\wedge X_2 = X_1 \sqcup I$
	$F_4 \leq maxfr$
$freq(X) - freq(X \sqcup \neg I) \leq \delta_2$	$\wedge \neg I \in X_4$
	$F_3 - F_4 \leq \delta_2$
	$\wedge X_4 = X_3 \sqcup \neg I$

The first four constraints correspond to the usual frequent classification. The last constraint expresses the overlapping between the premises: the two rules share at least half of the items of their premises. When a rule of this pair of rules is triggered by an unseen example, it means that it is likely that the other rule of the pair concluding to the another class value is also triggered and thus a classification conflict appears. The user can modify the parameters of the n-ary query and/or add new constraint to model specific classification conflicts.

4 CSP Modeling

We present the CSP modeling shared by the two approaches. Both of the approaches are based on two inter-related CSPs that we introduce in this section. The next section sketches the hybrid approach. Its main feature is to distinguish local constraints from n-ary ones. On the contrary, the full-CP approach does not make such a distinction, but all constraints will have to be reformulated using decision variables and reified constraints (see Section 6).

4.1 General Overview

Let \mathbf{r} be a dataset having m transactions, and \mathcal{I} the set of all its items. The itemset mining problem is modeled by using two inter-related CSPs \mathcal{P} and \mathcal{P}' :

1. Set CSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where:
 - $\mathcal{X} = \{X_1, \dots, X_n\}$. Each variable X_i represents an unknown itemset.
 - $\mathcal{D} = \{D_{X_1}, \dots, D_{X_n}\}$. Initial domain of X_i is the set interval $[\{\} .. \mathcal{I}]$.
 - \mathcal{C} is a conjunction of set constraints handling set operators (\cup, \cap, \in, \dots)
2. Numeric CSP $\mathcal{P}' = (\mathcal{F}, \mathcal{D}', \mathcal{C}')$ where:
 - $\mathcal{F} = \{F_1, \dots, F_n\}$. Each variable F_i is the frequency of the itemset X_i .
 - $\mathcal{D}' = \{D_{F_1}, \dots, D_{F_n}\}$. Initial domain of F_i is the integer interval $[1 .. m]$.
 - \mathcal{C}' is a conjunction of numeric constraints.

For each unknown itemset i , variables X_i and F_i are tied together (see Section 5.2 for the hybrid approach and Section 6.2 for the full-CP approach).

4.2 Example: Modeling the Exception Rules

Table 2 provides the primitive constraints for the exception rules n-ary query modelled in Section 3.1

- Set variables $\{X_1, X_2, X_3, X_4\}$ represent the four unknown itemsets:
 - $X_1 : X \setminus Y$, and $X_2 : (X \setminus Y) \sqcup I$ for the common sense rule,
 - $X_3 : X$, and $X_4 : X \sqcup \neg I$ for the exception rule.
- Integer variables $\{F_1, F_2, F_3, F_4\}$ represent their frequency values.
- Set constraints: $\mathcal{C} = \{(I \in X_2), (X_2 = X_1 \sqcup I), (\neg I \in X_4), (X_4 = X_3 \sqcup \neg I), (X_1 \subsetneq X_3)\}$
- Numeric constraints: $\mathcal{C}' = \{(F_2 \geq \text{minfr}), (F_1 - F_2 \leq \delta_1), (F_4 \leq \text{maxfr}), (F_3 - F_4 \leq \delta_2)\}$

5 Related Work

5.1 Pattern Discovery Approaches in Data Mining

There are a lot of works to discover local patterns under constraints [7,15] but there are no so many methods to combine local patterns. Recent approaches - pattern teams [14], constraint-based pattern set mining [9] and selecting patterns according to the added value of a new pattern given the currently selected patterns [5] - aim at reducing the redundancy by selecting patterns from the initial large set of local patterns on the basis of their usefulness in the context of the other selected patterns. Even if these approaches explicitly compare patterns between them, they are mainly based on the reduction of the redundancy or specific aims such as classification processes. We argue that n-ary queries are a flexible way to take into account a bias given by the user to direct the final set of patterns toward a specific aim such as the search of exceptions. General data mining frameworks based on the notion of local patterns to design global models are presented in [13,11]. These frameworks help to analyze and improve current methods in the area.

5.2 A Hybrid Method

We did a preliminary work [12] based on a joint use of local patterns constraint-based mining and CP in order to discover n-ary patterns and to investigate such relationships. In this section, we provide an overview of the method highlighted by the example of the exception rules n-ary query (see Section 4.2).

i) General overview. The hybrid approach consists in three steps:

1. Modeling the n-ary query as a CSP, then splitting constraints into local ones and n-ary ones.
2. Solving the local constraints using a local patterns extractor (MUSIC-DFS [2] [18] which produces an interval condensed representation of all patterns satisfying the local constraints.

² <http://www.info.univ-tours.fr/~soulet/music-dfs/music-dfs.html>

3. Solving the n-ary constraints using ECL^iPS^e ³. The domain of each variable results from the interval condensed representation (computed in the Step-2).

ii) Splitting the set of constraints. The whole set of constraints ($\mathcal{C} \cup \mathcal{C}'$) is split into two subsets as follows:

- \mathcal{C}_{loc} is the set of local constraints to be solved (by MUSIC-DFS). Solutions are given in the form of an interval condensed representation.
- \mathcal{C}_n is the set of n-ary constraints to be solved where the domains of the variables X_i and F_i will be deduced from the interval condensed representation.

For the exception rules (see Table 2) this splitting is performed as follows:

- $\mathcal{C}_{loc} = \{(I \in X_2), (F_2 \geq minfr), (F_4 \leq maxfr), (\neg I \in X_4)\}$
- $\mathcal{C}_n = \{(F_1 - F_2 \leq \delta_1), (X_2 = X_1 \sqcup I), (F_3 - F_4 \leq \delta_2), (X_4 = X_3 \sqcup \neg I), (X_1 \subsetneq X_3)\}$

iii) Solving the local constraints. MUSIC-DFS is a local patterns extractor which offers a set of syntactic and aggregate primitives to specify a broad spectrum of local constraints in a flexible way [18]. MUSIC-DFS mines soundly and completely all the patterns satisfying a given set of local constraints. The local patterns satisfying all the local constraints are provided in a condensed representation made of intervals (each interval represents a set of patterns satisfying the constraint and each pattern appears in only one interval. The lower bound of an interval is a prefix-free pattern and its upper bound is the prefix-closure of the lower bound [18]).

For the hybrid method, local constraints are solved before and regardless n-ary constraints. So that, the search space for n-ary constraints will be reduced to the space of solutions of local constraints. The set of local constraints \mathcal{C}_{loc} is split into a disjoint union of \mathcal{C}_i (for $i \in [1..n]$) where each \mathcal{C}_i is the set of local constraints related to X_i and F_i . Each \mathcal{C}_i can be separately solved. Let CR_i be the interval condensed representation of all the solutions of \mathcal{C}_i . $CR_i = \bigcup_p (f_p, I_p)$ where I_p is a set interval verifying: $\forall x \in I_p, freq(x) = f_p$.

So domains for variable X_i and variable F_i are:

- D_{F_i} : the set of all f_p in CR_i
- D_{X_i} : $\bigcup_{I_p \in CR_i} I_p$

Example: Let us consider \mathcal{C}_{loc} the set of local constraints for the exception rules. The respective values for $(I, \neg I, minfr, \delta_1, maxfr, \delta_2)$ are $(c_1, c_2, 2, 1, 1, 0)$. The set of local constraints related to X_2 and F_2 , $\mathcal{C}_2 = \{c_1 \in X_2, F_2 \geq 2\}$, is solved by MUSIC-DFS with the following query showing that the parameters can be straightforwardly deduced from \mathcal{C}_{loc} .

```

-----
./music-dfs -i donn.bin -q "{c1} subset X2 and freq(X2)>=2;"
X2 in [A, c1]..[A, c1, B ] U [B, c1] -- F2 = 2 ;
X2 in [C, c1] -- F2 = 3
-----

```

iv) Solving the n-ary constraints. Domains of the variables X_i and F_i are deduced from the condensed representation of all patterns satisfying local constraints. Solving n-ary constraints using ECL^iPS^e enables then to obtain all the solutions satisfying the whole set of constraints (local ones and n-ary ones).

³ <http://www.eclipse-clp.org>

Example: Let us consider \mathcal{C}_n the set of n-ary constraints for the exception rules. The respective values for $(I, \neg I, \text{minfr}, \delta_1, \text{maxfr}, \delta_2)$ are still the same. The following *ECLⁱPS^e* session illustrates how all pairs of exception rules can be directly obtained by using backtracking:

```

-----
[eclipse 1]:
?- exceptions(X1, X2, X3, X4).
Sol1 : X1 = [A,B], X2=[A,B,c1], X3=[A,B,C], X4=[A,B,C,c2];
Sol2 : X1 = [A,B], X2=[A,B,c1], X3=[A,B,D], X4=[A,B,D,c2];
.../...
-----

```

6 A Full-CP Approach

This section presents a new approach keeping the previous modeling (see Section 4) but using a new solving technique. We call *full-CP* this approach because it only uses a CP solver and no longer a local patterns extractor. Section 7 analyzes in depth the two methods.

Numeric constraints and set constraints are modeled in three steps: linking the data and the patterns involved in the n-ary constraint, then modeling the patterns and finally formulating numeric constraints and set constraints.

We use the CP system Gecode⁴. For the first step (see Section 6.1), we use the implementation of the Itemset Mining system FIM_CP⁵ which is an approach using CP for pattern mining [8]. This approach addresses in a unified framework a large set of local patterns and constraints such as frequency, closedness, maximality, constraints that are monotonic or anti-monotonic or variations of these constraints but it does not deal with n-ary constraints.

In the remainder of this section, let \mathbf{r} be a dataset where \mathcal{I} is the set of its n items and \mathcal{T} the set of its m transactions, and let \mathbf{d} be the 0/1 (m, n) matrix where $\forall t \in \mathcal{T}, \forall i \in \mathcal{I}, (d_{t,i} = 1) \Leftrightarrow (i \in t)$.

6.1 Modeling an Unknown Local Pattern

Let M be the unknown local pattern. FIM_CP establishes the link between the data set and M by introducing two kinds of variables, each of them having $\{0, 1\}$ for domain: $\{M_1, M_2, \dots, M_n\}$ where $(M_i = 1) \Leftrightarrow (i \in M)$, and $\{T_1, T_2, \dots, T_m\}$ where $(T_t = 1) \Leftrightarrow (M \subseteq t)$. So, $\text{freq}(M) = \sum_{t \in \mathcal{T}} T_t$ and $\text{length}(M) = \sum_{i \in \mathcal{I}} M_i$.

The relationship between M and \mathcal{T} is modeled by reified constraints, stating that, for each transaction t , $(T_t = 1)$ iff t is covered by M :

$$\forall t \in \mathcal{T}, (T_t = 1) \Leftrightarrow \sum_{i \in \mathcal{I}} M_i \times (1 - d_{t,i}) = 0 \quad (1)$$

⁴ <http://www.gecode.org>

⁵ http://www.cs.kuleuven.be/~dtai/CP4IM/fim_cp.php

Each reified constraint (see Equation 3) is solved as follows: if the solver deduces $(T_t=1)$ (resp. $T_t=0$), then the sum must be equal to 0 (resp. must be different from 0). The propagation is also performed, in a same way, from the sum constraint toward the equality constraint.

6.2 Modeling the k Patterns We Are Looking for

Let X_1, X_2, \dots, X_k be the k patterns we are looking for. First, each pattern X_j is modeled by n variables $\{X_{1,j}, X_{2,j}, \dots, X_{n,j}\}$ having $\{0, 1\}$ for domain and such that $(X_{i,j} = 1)$ iff item i belongs to pattern X_j :

$$\forall i \in \mathcal{I}, (X_{i,j} = 1) \Leftrightarrow (i \in X_j) \tag{2}$$

m variables $\{T_{1,j}, T_{2,j}, \dots, T_{m,j}\}$ having $\{0, 1\}$ for domain are associated with each pattern X_j such that $(T_{t,j} = 1)$ iff transaction t is covered by pattern X_j :

$$\forall t \in \mathcal{T}, (T_{t,j} = 1) \Leftrightarrow (X_j \subseteq t) \tag{3}$$

So, $freq(X_j) = \sum_{t \in \mathcal{T}} T_{t,j}$ and $length(X_j) = \sum_{i \in \mathcal{I}} X_{i,j}$. The relationship between each pattern X_j and \mathcal{T} is modeled by reified sum constraints, stating that, for each transaction t , $(T_{t,j} = 1)$ iff t is covered by X_j :

$$\forall j \in [1..k], \forall t \in \mathcal{T}, (T_{t,j} = 1) \Leftrightarrow \sum_{i \in \mathcal{I}} X_{i,j} \times (1 - d_{t,i}) = 0 \tag{4}$$

6.3 Reformulating Numeric and Set Constraints

Let operator $op \in \{<, \leq, >, \geq, =, \neq\}$; numeric constraints are reformulated as follows:

- $freq(X_p) \text{ op } \alpha \rightarrow \sum_{t \in \mathcal{T}} T_{t,p} \text{ op } \alpha$
- $length(X_p) \text{ op } \alpha \rightarrow \sum_{i \in \mathcal{I}} X_{i,p} \text{ op } \alpha$

Some set constraints (such that equality, inclusion, membership) are directly reformulated using linear numeric constraints:

- $X_p = X_q \rightarrow \forall i \in \mathcal{I}, X_{i,p} = X_{i,q}$
- $X_p \subseteq X_q \rightarrow \forall i \in \mathcal{I}, X_{i,p} \leq X_{i,q}$
- $i_o \in X_p \rightarrow X_{i_o,p} = 1$

Other set constraints (such that intersection, union, difference) can easily be reformulated into boolean constraints using the conversion function $(b :: \{0, 1\} \rightarrow \{False, True\})$ and the usual boolean operators:

- $X_p \cap X_q = X_r \rightarrow \forall i \in \mathcal{I}, b(X_{i,r}) = b(X_{i,p}) \wedge b(X_{i,q})$
- $X_p \cup X_q = X_r \rightarrow \forall i \in \mathcal{I}, b(X_{i,r}) = b(X_{i,p}) \vee b(X_{i,q})$
- $X_p \setminus X_q = X_r \rightarrow \forall i \in \mathcal{I}, b(X_{i,r}) = b(X_{i,p}) \wedge \neg b(X_{i,q})$

Finally, reified constraints (linking the dataset and the patterns, see Equation 4), numeric constraints and set constraints are all managed by *Gecode*.

6.4 Experiments

This section shows the practical usage and the feasibility of our approach.

i) Experimental setup. Experiments were performed on several datasets from the UCI repository⁶ and a real-world dataset **Meningitis** coming from the Grenoble Central Hospital⁷. This last dataset gathers children hospitalized for bacterial or viral meningitis. Table 3 summarizes the characteristics of these datasets. Experiments were conducted with several kinds of n-ary queries: exception rules, unexpected rules and classification conflicts. We use a PC having a 2.83 GHz Intel Core 2 Duo processor and 4 GB of RAM, running Ubuntu Linux.

Table 3. Description of the datasets

dataset	#trans	#items	density
Mushroom	8142	117	0.18
Australian	690	55	0.25
Meningitis	329	84	0.27

ii) Soundness and Flexibility. As the resolution performed by the CP solver is sound and complete, our approach is able to mine the correct and complete set of patterns satisfying n-ary queries. Figure 1 (the upper part) depicts the number of pairs of exception rules according to *minfr* and δ_1 and Figure 1 (the bottom part) indicates the number of classification conflict rules according to the two parameters *minfr* and *minconf*. These figures show the feasibility of our approach that mines the correct and complete set of all patterns satisfying the n-ary queries from these various sets of parameters. We tested other combinations of the parameters: as they provided similar results, they are not indicated here. As expected, the lower *minfr* is, the larger the number of pairs of exception rules. Results are similar when δ_1 varies: the higher δ_1 is, the larger the number of pairs of exception rules (when δ_1 increases, the confidence decreases so that there are more common sense rules).

iii) Highlighting useful patterns. As already said, exception rules are a particular case of rare rules (cf. Section 2.2). There are few attempts to extract the whole set of rare rules [20]. But, even if these rules can be extracted, it is impossible to pick the exception rules among the set of all the rare rules. That is a strong limitation because most of the rare rules are unreliable and it highlights the interest of pairs of exceptions rules. Figure 2 quantifies the number of pairs of exception rules on the **Meningitis** dataset versus the number of rare rules (the number of rare rules which depends on *maxfr* corresponds to the line at the top of the figure). Looking for pairs of exception rules reduces on several orders of magnitude the number of outputted patterns (the Y-axis is on a logarithmic scale). A n-ary constraint enables to straightforwardly discover the proper set of exception rules.

⁶ <http://www.ics.uci.edu/~mlearn/MLRepository.html>

⁷ The authors would like to thank Dr P. François who provided the meningitis dataset.

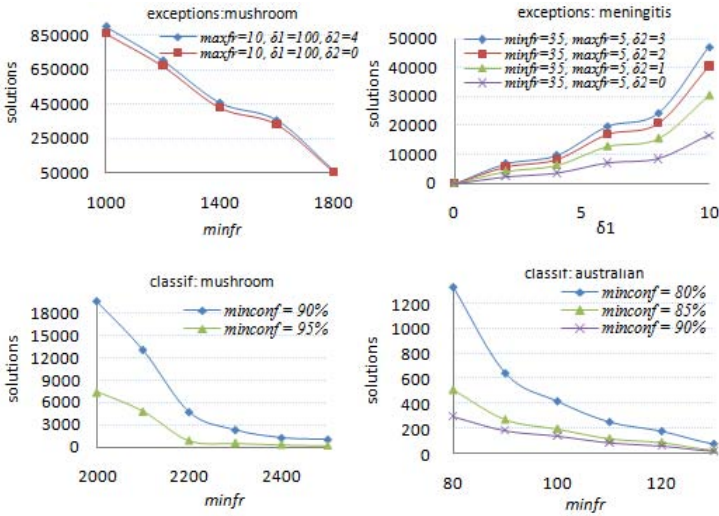


Fig. 1. Number of solutions

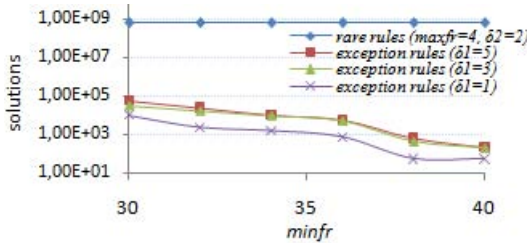


Fig. 2. Number of pairs of exception rules versus number of rare rules (Meningitis)

Unexpected rules may reveal useful information. For example, on *Meningitis* dataset, such a rule has a premise made of a high percentage of immature band cells and the absence of neurological deficiency and its conclusion is a normal value of the polynuclear neutrophil level. This rule is unexpected with the belief that high values of the white cells count and the polynuclear percentage lead to a bacterial etiological type. Experts appreciate to have n-ary constraints to address such patterns.

iv) Computational Efficiency. This experiment quantify runtimes and the scalability of our approach. In practice, runtimes vary according to the size of the datasets and the tightness of constraints (a constraint is said *tight* if its number of solutions is low compared to the cardinality of the cartesian product of the variable domains, such as constraints defined by high frequency and confidence thresholds).

For *Meningitis* and *Australian*, the set of all solutions is computed in a few seconds (less than one second in most of the cases). On *Mushroom*, runtimes

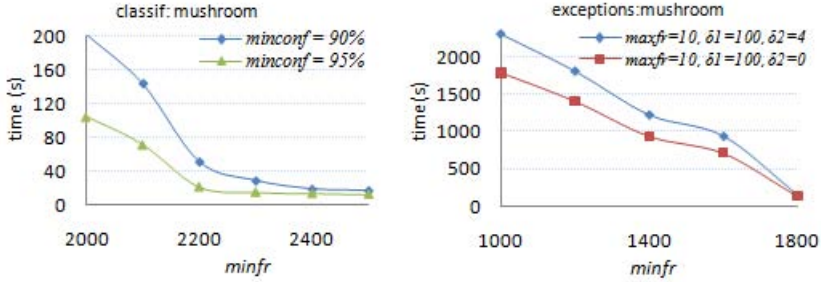


Fig. 3. Runtimes

vary from few seconds for tight constraints to about an hour for low frequency and confidence thresholds. So we have conducted further experiments on this dataset. Figure 3 details the runtime of our method on Mushroom according to different thresholds of confidence and frequency. We observe that the tighter the constraint is, the smaller the runtime is. Indeed, tight constraints enable a better filtering of the domains and then a more efficient pruning of the search tree. Runtimes also depend on the size of the dataset: the larger the dataset is, the larger the resulting CP program (cf. Section 6).

Obviously, our generic n-ary approach can be used for mining local patterns. We obtain on this task the same runtimes as [8] which were competitive with state of the art miners. With exception rules, we cannot compare runtimes because they are not indicated in [19].

7 Discussion

The hybrid approach (see Section 5.2) and the full-CP approach proposed in this paper (see Section 6) follow the same modeling described in Section 4. But, they have different solving methods. The hybrid approach uses a local patterns extractor in order to produce an interval condensed representation (of all patterns satisfying the local constraints) that will be used for constituting the domains of the Set CSP variables. The full-CP approach only uses a CP solver and no longer a local patterns extractor.

7.1 Hybrid Approach: Pros and Cons

With the hybrid approach, the modeling of the n-ary query can be directly provided to a Set CSP solver without any reformulation. Thus, a prototype can quickly be developed by using a Set CSP solver such as *ECLⁱPS^e*.

But, Set CSP solvers [10] do not well manage the union of set intervals. In order to establish bound consistency, the union of two set intervals is approximated by its convex closure [8]. To circumvent this problem, for each variable

⁸ The convex closure of $[lb_1 .. ub_1]$ and $[lb_2 .. ub_2]$ is defined as $[lb_1 \cap lb_2 .. ub_1 \cup ub_2]$.

X_i with the condensed representation $CR_i = \bigcup_p (f_p, I_p)$, a search is successively performed upon each I_p . If this approach is sound and complete, it does not fully profit from filtering because value removals are propagated only in the handled intervals and not in the whole domains.

This fact could seem to be prohibitive, but the number of set intervals strongly decreases according to local constraints. Table 4 indicates the number of set intervals constituting the domain of variable X_2 according to several local constraints (see the exception rules example Section 5.2).

An alternative solution would be to use non-exact condensed representations to reduce the number of produced intervals (e.g., a condensed representation based on maximal frequent itemsets [6]). In this case, the number of intervals representing the domains will be rather small, but, due to the approximations, it should be necessary to memorize forbidden values.

7.2 Full-CP Approach: Pros and Cons

First, the reformulation of n-ary constraints to low level constraints can be performed in an automatic way. Moreover, the filtering of reified constraints performed by *Gecode* is very effective and performant. But, the resulting number of constraints must be high for very large datasets. Let us consider a dataset with n items and m transactions and a n-ary query involving k unknown patterns. Linking the dataset and the unknown patterns requires $(k \times m)$ constraints, each of them involving at most $(n+1)$ variables (see Equation 4 in Section 6.2). So, the number of constraints necessary to model very large datasets could be prohibitive. In practice, this approach is able to tackle a large set of databases.

To summarize. The two approaches mainly differ in the way they consider the whole dataset. The hybrid approach uses a local patterns extractor and the resulting CSP owns a very small number of constraints, but variables with large domains. On the other hand, the full-CP approach requires a large number of constraints over decision variables. Experiments show that the full-CP approach is significantly more performant than the hybrid one on large datasets. Ratios between their respective runtimes are up to several orders of magnitude. This is due to the lower quality of the filtering of the hybrid approach (see Section 7.1).

Table 4. Number of intervals according to several local constraints (case of D_{X_2})

Local constraint	Number of intervals in D_{X_2}
-	3002
$I \in X_2$	1029
$I \in X_2 \wedge freq(X_2) \geq 20$	52
$I \in X_2 \wedge freq(X_2) \geq 25$	32

8 Conclusion and Future Works

In this paper, we have presented an approach to model and mine n-ary patterns. To the best of our knowledge, it is the first generic approach to express and mine patterns involving several local patterns. The examples described in Section 3 illustrate the generality and the flexibility of our approach. Experiments show its relevance and its feasibility in spite of its generic scope.

For CSPs, all variables are existentially quantified. Further work is to introduce the universal quantification: this quantifier would be precious to model important constraints such as the *peak* constraint (the *peak* constraint compares neighbor patterns; a *peak* pattern is a pattern whose all neighbors have a value for a measure lower than a threshold). For that purpose, we think that Quantified CSPs [14] could be appropriate and useful.

On the other hand, extracting actionable and interesting knowledge from data is a human-guided, iterative and interactive process. The data analyst should only consider a high-level vision of the pattern discovery system and handle a set of primitives to declaratively specify the pattern discovery task. Even if CP is a new approach to tackle this problem [8,12,16], it appears to be very promising for building such a high level and interactive system.

References

1. Benedetti, M., Lallouet, A., Vautard, J.: Quantified constraint optimization. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 463–477. Springer, Heidelberg (2008)
2. Besson, J., Robardet, C., Boulicaut, J.-F.: Mining a new fault-tolerant pattern type as an alternative to formal concept discovery. In: ICCS 2006, Aalborg, Denmark, pp. 144–157. Springer, Heidelberg (2006)
3. Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., Trasarti, R.: A constraint-based querying system for exploratory pattern discovery. *Inf. Syst.* 34(1), 3–27 (2009)
4. Bordeaux, L., Monfroy, E.: Beyond NP: Arc-consistency for quantified constraints. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 371–386. Springer, Heidelberg (2002)
5. Bringmann, B., Zimmermann, A.: The chosen few: On identifying valuable patterns. In: 12th IEEE Int. Conf. on Data Mining (ICDM 2007), pp. 63–72 (2007)
6. Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., Yiu, T.: Mafia: A performance study of mining maximal frequent itemsets. In: FIMI 2003. CEUR Workshop Proceedings, vol. 90, CEUR-WS.org (2003)
7. Calders, T., Rigotti, C., Boulicaut, J.-F.: A survey on condensed representations for frequent sets. In: Boulicaut, J.-F., De Raedt, L., Mannila, H. (eds.) Constraint-Based Mining and Inductive Databases. LNCS (LNAI), vol. 3848, pp. 64–80. Springer, Heidelberg (2006)
8. De Raedt, L., Guns, T., Nijssen, S.: Constraint Programming for Itemset Mining. In: ACM SIGKDD Int. Conf. KDD 2008, Las Vegas, Nevada, USA (2008)
9. De Raedt, L., Zimmermann, A.: Constraint-based pattern set mining. In: 7th SIAM Int. Conf. on Data Mining. SIAM, Philadelphia (2007)

10. Gervet, C.: Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *Constraints* 1(3), 191–244 (1997)
11. Giacometti, A., Khanjari Miyaneh, E., Marcel, P., Soulet, A.: A framework for pattern-based global models. In: Corchado, E., Yin, H. (eds.) *IDEAL 2009*. LNCS, vol. 5788, pp. 433–440. Springer, Heidelberg (2009)
12. Khiari, M., Boizumault, P., Crémilleux, B.: Combining CSP and constraint-based mining for pattern discovery. In: Taniar, D., Gervasi, O., Murgante, B., Pardede, E., Apduhan, B.O. (eds.) *ICCSA 2010*. LNCS, vol. 6017, pp. 432–447. Springer, Heidelberg (2010)
13. Knobbe, A., Crémilleux, B., Fürnkranz, J., Scholz, M.: From local patterns to global models: The lego approach to data mining. In: *Int. Workshop LeGo collocated with ECML/PKDD 2008*, Antwerp, Belgium, pp. 1–16 (2008)
14. Knobbe, A., Ho, E.: Pattern teams. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) *PKDD 2006*. LNCS (LNAI), vol. 4213, pp. 577–584. Springer, Heidelberg (2006)
15. Ng, R.T., Lakshmanan, V.S., Han, J., Pang, A.: Exploratory mining and pruning optimizations of constrained associations rules. In: *Proceedings of ACM SIGMOD 1998*, pp. 13–24. ACM Press, New York (1998)
16. Nijssen, S., Guns, T., De Raedt, L.: Correlated itemset mining in ROC space: a constraint programming approach. In: *KDD 2009*, pp. 647–655 (2009)
17. Padmanabhan, B., Tuzhilin, A.: A belief-driven method for discovering unexpected patterns. In: *KDD*, pp. 94–100 (1998)
18. Soulet, A., Klema, J., Crémilleux, B.: Efficient mining under rich constraints derived from various datasets. In: Dzeroski, S., Struyf, J. (eds.) *KDID 2006*. LNCS, vol. 4747, pp. 223–239. Springer, Heidelberg (2007)
19. Suzuki, E.: Undirected Discovery of Interesting Exception Rules. *Int. Journal of Pattern Recognition and Artificial Intelligence* 16(8), 1065–1086 (2002)
20. Szathmary, L., Napoli, A., Valtchev, P.: Towards Rare Itemset Mining. In: *Proc. of the 19th IEEE ICTAI 2007*, Patras, Greece, vol. 1 (2007)
21. Yin, X., Han, J.: CPAR: classification based on predictive association rules. In: *Proceedings of the 2003 SIAM Int. Conf. on Data Mining, SDM 2003* (2003)

An Integrated Business Rules and Constraints Approach to Data Centre Capacity Management

Roman van der Krogt, Jacob Feldman, James Little, and David Stynes

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland

Abstract. A recurring problem in data centres is that the constantly changing workload is not proportionally distributed over the available servers. Some resources may lay idle while others are pushed to the limits of their capacity. This in turn leads to decreased response times on the overloaded servers, a situation that the data centre provider wants to prevent. To solve this problem, an administrator may move (reallocate) applications or even entire virtual servers around in order to spread the load. Since there is a cost associated with moving applications (in the form of down time during the move, for example), we are interested in solutions with minimal changes. This paper describes a hybrid approach to solving such resource reallocation problems in data centres, where two technologies have to work closely together to solve this problem in an efficient manner.

The first technology is a Business Rules Management System (BRMS), which is used to identify which systems are considered to be overloaded on a systematic basis. Data centres use complex rules to track the behaviour of the servers over time, in order to properly identify overloads. Representing these tracking conditions is what the BRMS is good for. It defines the relationships (business constraints) over time between different applications, processes and required resources that are specific to the data centre. As such, it also allows a high degree of customisation.

Having identified which servers require reallocation of their processes, the BRMS then automatically creates an optimisation model solved with a Constraint Programming (CP) approach. A CP solver finds a feasible or the optimal solution to this CSP, which is used to provide recommendations on which workload should be moved and whereto. Notice that our use of a hybrid approach is a requirement, not a feature: employing only rules we would not be able to compute an optimal solution; using only CP we would not be able to specify the complex identification rules without hard-coding them into the program. Moreover, the dedicated rule engine allows us to process the large amounts of data rapidly.

1 Introduction

Data centres are “*buildings where multiple servers and communication gear are colocated because of their common environmental requirements and physical security needs, and for ease of maintenance*” [1]. They have become increasingly

more important due to two trends. Firstly, there is a need to process larger and larger workloads and store ever increasing amounts of data. An obvious example of this would be search engines on the World Wide Web, but also consider supermarkets that use loyalty cards to track what their customers buy. More companies storing and analysing larger amounts of data means that an increased amount of computing power is needed to satisfy the demand. At the same time, computing and storage is moving from PCs to internet services and so more centralised storage and computing power is required. Besides reasons of scale and ease of management (such as centralised backups), there is also a large economical reason: applications run at a lower cost per user, as servers may be shared by many active users and many more inactive ones. Virtualisation is the latest trend in this area. One can now acquire virtual servers, that scale with the computing resources required (*cloud computing*). Many such virtual servers can reside on the same physical server, providing low cost solutions.

However, data centres also come with new computational challenges. The problem we will focus on in this paper is due to the dynamic nature of the environment. Most processes behave in a very erratic way, switching from using few resources to many and back.¹ For example, consider a mail server. This remains dormant until email arrives at which point it wakes up and processes the email. One can have many dormant processes on a server, but only a limited number of active ones. When too many are active, the response times decrease and the service level goes down. This results in the appearance of an application running slow and hence affects the user experience.

Similar concerns arise in cloud computing. Many small virtual servers can be hosted on a single physical server, but when too many want to scale up simultaneously (whether in terms of CPU capacity, available memory, bandwidth or otherwise) there is simply not enough room to accommodate all of them. Such situations are currently dealt with through human operators. From their control room, they are able to see the characteristics of the servers (including the current CPU load, memory and bandwidth consumption, and possibly including factors such as power usage and temperature). When they recognise unwanted patterns (e.g. a server being overloaded for a certain period) they will investigate and manually solve the problem by moving processes around. The current state-of-the-art support tools for data centre administrators (as, e.g. the CapacityAnalyzer product of VKernel [\[2\]](#)) provide them not only with a monitoring tool, but can also diagnose or even predict problems and bottlenecks, i.e. indicate which servers are running critical. However, the tools do not provide a recommendation as to which process should be moved, or whereto. The system we describe in this paper is a comprehensive system, that

1. Acts as an early warning system for potential problems with the servers in the data centre; and

¹ For convenience, we will use the term “process” to refer to the particular objects that we can reallocate between servers. These could be processes in the traditional sense, but in different circumstances, it could mean a complete virtual server consisting of many traditional processes.

2. Proposes solutions in the event problems are indeed detected.

The problem naturally decomposes into two separate subproblems of *(i)* problem identification and formulation and *(ii)* problem resolution. Correspondingly, the proposed system uses two different solution techniques, each most suited to the particular subproblem. We will use a Business Rule Management System (BRMS) to analyse the information stream that is provided by the monitoring software. To this end, we define in the BRMS the relationships between the different applications, processes and required resources. Using this information, the BRMS can efficiently track the state of the servers over time. By applying business rules, it identifies resource overloading situations and potential bottlenecks. If there are any issues detected, we will formulate an appropriate resource reallocation problem to be solved using Constraint Programming (CP). The purpose of the CP formulation is to find a set of minimal changes to the current configuration in order to resolve the identified problem. (For example, in the case of a CPU overload, a busy application could be moved from the server to one with a low level of CPU utilisation.) As different types of data centres prefer different kinds of reallocations, the user can specify their objectives through the BRMS and have them taken into account when generating the CSP.

A key contribution of our work is the fact that the CP formulation is automatically derived through the rule engine. It puts the business specialists (i.e. data centre operators) in charge of the workload re-configuration process allowing them to concentrate on what they want to achieve, instead of how this should be achieved. Existing combinations of BRMS and optimisation (which we will discuss in a later section) have used rules to provide the input to the optimisation problem, and/or use rules to direct the search. In contrast, the entire problem formulation is driven by the rules in our approach. The rules that are triggered will add variables and constraints to the model, thus constructing it in real-time, based on the actual, data centre-specific situation. A second contribution is derived from the integration of CP within Business Rules, which allows people with no expert knowledge of optimisation in general, or CP in particular, to express and change their problem simply through the rule editor. This makes it easier for business analysts (to whom the modern business rules systems are oriented) to maintain CP models.

The remainder of the paper is organised as follows. First, we provide a formal problem specification. Then, in Section 3, we detail the proposed solution. Section 4 provides an experimental analysis of our approach. After discussing related work, we draw conclusions and provide pointers for future work.

2 Formal Problem Specification

Our data centre consists of a set $S = \langle s_1, \dots, s_n \rangle$ of n servers. Depending on the type of data centre, these servers can be used to provide computing power, data storage, a combination of these, or any other purpose. This is of no consequence to the model itself. Furthermore, there is a set $P = \langle p_1, \dots, p_m \rangle$ of processes. At regular time intervals, a set of l sensors provide information on the processes,

e.g. their CPU load, the amount of memory required or energy consumption. A function $\sigma_i^t : P \rightarrow \mathbb{R} \cup \{\perp\}$, $i \in [0, l]$ provides the output of a sensor for a given time point and a given process, i.e. $\sigma_i^t(p)$ equals the value measured by sensor i at time t for the process p . If a process is not active at a certain time point t (e.g. because it has finished), the output of σ_i^t , $i \in [0, l]$ is undefined, i.e. $\sigma_i^t = \perp$. By default, sensor 0 returns the server that the process is running on, i.e. $\sigma_o^t(p) = j$ iff p runs on s_j at time t .

The total requirements placed on a server s_j can be computed from the processes running on that server as follows:

$$\sigma_i^t(s_j) = \sum_{\{p \mid \sigma_o^t(p)=j\}} \sigma_i^t(p) \quad i \in [1, l]$$

We will use $\sigma^t(s) = \langle \sigma_0^t(s), \dots, \sigma_l^t(s) \rangle$ to denote the set of values of all sensors for a given server at a particular time. We let Σ denote the set of all possible combinations of sensor readings, i.e. $\sigma^t(s) \in \Sigma$.

To identify which servers are of interest, we introduce a classification \mathbb{S} of possible labels (for example, this could be equal to $\mathbb{S} = \{\text{critical, high, medium, low}\}$). We assume there is a function $\Delta : S \times \Sigma^z \rightarrow \mathbb{S}$ that, given a set of sensor readings for the past z time steps, can give the state a server is in. In order to optimise our solution, we introduce a cost function $cost : \mathbb{S} \rightarrow \mathbb{R}^+$ that, given a label, returns a virtual cost value for that particular label.

Through the classification of servers, we may identify that an unwanted situation has arisen. For example, we may find a number of servers that are classified as critical. To address this issue, we could move processes away from these servers onto others. Thus, a solution is a reallocation of processes that brings the data centre to an improved state. In terms of the model, we want to compute a set $\{\sigma_o^{t+1}(p) \mid p \in P\}$.

There are a number of constraints that should hold for a solution. Firstly, there is a maximum value max_i that we want to satisfy for each sensor $\sigma_i^t(s)$ of a server s . Secondly, some processes may never run together, i.e. be present on the same server. To this end, there is a function $incompatible : P \rightarrow 2^P$ that given some process returns the list of all processes that cannot share the same resources with that process.² Finally, some processes may need a particular set of servers. The function $possible : P \rightarrow 2^S$ denotes the possible servers that a process can run on. (Thus, $possible(p) = S$ if there are no restrictions on p .)

The last aspect we introduce is a distance metric between servers. The function $\delta : S \times S \rightarrow \mathbb{R}^+$ returns the distance between two servers. This could simply be the physical distance, but is more likely to also take into account the network

² A possible reason for such an incompatibility constraint may be that a critical application is distributed over multiple servers to provide a level of fault tolerance. We should then prevent that two instances of this application are running on the same server, in order to guarantee the application is still available when the server crashes. Security aspects are another typical example.

topology in the data centre. When moving processes, we want to minimise the total distance of the processes that are moved. Note that, in general,

$$\min_{s \in S} cost(s) > \max_{s_1, s_2 \in S} \delta(s_1, s_2) \tag{1}$$

That is, our first priority is in achieving acceptable levels for the states of the servers, with minimum movements of secondary importance.³

The full problem statement can now be formalised as follows.

Given $S = \langle s_1, \dots, s_n \rangle$ servers
 $P = \langle p_1, \dots, p_m \rangle$ processes
 $\sigma^0, \dots, \sigma^t$ sensor readings, where $\sigma^i = \langle \sigma_0^i, \dots, \sigma_l^i \rangle$

Find $\sigma_o^{t+1}(p)$ for each $p \in P$

Subject to $\forall p \in P \cdot \sigma_i^{t+1}(p) = \sigma_i^t(p), i = [1, l]$

$\forall s \in S \cdot \sigma_i^{t+1}(s) \leq max_i$

$\forall p \in P \cdot \sigma_0^{t+1}(p) \in possible(p)$

$\forall p_1, p_2 \in P \cdot p_1 \in incompatible(p_2) \implies \sigma_o^{t+1}(p_1) \neq \sigma_o^{t+1}(p_2)$

Minimising $\sum_{s \in S} cost(\Delta(s, \sigma^{t+1}(s), \dots, \sigma^{t+1-z}(s)))$

$$+ \sum_{p \in P} \delta(\sigma_0^t(p), \sigma_0^{t+1}(p))$$

Thus, we are interested in finding a new allocation of processes to servers for the next time step, i.e. computing $\sigma_o^{t+1}(p)$ for all $p \in P$, such that the allocation satisfies all constraints. The cost of this allocation is mainly influenced by the cost associated with the labelling of the servers. In the next section, we introduce an approach that efficiently computes the labelling and solves the problem.

3 A Hybrid Approach

Our approach to solve the problem naturally breaks down into three stages:

1. The identification of servers which are at the different risk levels;
2. The formulation of the re-distribution problem; and
3. Solving the generated problem to compute a new workload allocation with as few changes as possible.

3.1 Stage 1a: Rules-Based Problem Identification

The first stage relates to the implementation of the function Δ that we introduced in the previous section. The problem here is that this function may be

³ However, note that $cost(s)$ might equal $cost(s) = 0$, in which case these states should be disregarded in Equation \square

very complex due to the many conditions over time that need to be taken into account. The biggest cause of this complexity is a desire for stability. It would be too costly to reallocate processes each time a certain sensor reading is too high and therefore, we need to establish a pattern emerging over time before we decide to take action. There is also a second issue: flexibility. This is desired because not all variations of the problem are known ahead of time, and this allows a user to customise for their situation: the SLAs they have agreed with users, standard procedures, etc.

Most data centres already use monitoring systems that provide performance, resource utilisation and workload projection capabilities. The resource utilisation data usually contains a number of samples taken over monitored time intervals that range from 30 seconds to 30 minutes. This data indicates when and how frequently different utilisation spikes occur. Business rules are an ideal way to formulate the process of analysing the monitoring data and define spike frequencies and overload thresholds.

For the purpose of this research, we consider a business rule to be a triplet $b_\chi = \langle e, c, a \rangle$ [3], where χ is a list of variables, e describes an event that triggers the rule, c describes a condition over χ that has to be met, and a specifies the action to perform. Rules are grouped in *rule sets*. When an event happens, the rules in each applicable rule set $B = \{b_1, \dots, b_r\}$ are evaluated in order, starting from b_1 , and identifying the actual values with the variables χ . Depending on the strategy the BRMS employs, one or more of the matching rules is executed. Executing multiple matching rules allows more specific conditions to override generic ones. This is the strategy taken by our system.

By enumerating the different cases described by Δ we can capture the behaviour of Δ in a rule set. The trigger event for each rule is the update in the sensor readings; the condition describes the case to which it applies; and the action specifies the label that Δ outputs for that particular case. For example, consider the following very simple labeling function, with the number of timesteps equal to $z = 1$, $\mathbb{S} = \{\text{high, medium, low}\}$, and assuming $\sigma_1^t \in [0, 100]$ measures the CPU load in percentages:

$$\Delta(s, \sigma) = \begin{cases} \text{low} & \text{if } \sigma_1(s) \leq 60 \\ \text{medium} & \text{if } 60 < \sigma_1(s) \leq 90 \\ \text{high} & \text{if } 90 < \sigma_1(s) \leq 100 \end{cases}$$

This can be efficiently represented using the rule set of Table 1, where ϵ denotes an update to the sensor readings. More complicated functions can be represented by rule sets in a similar fashion; for example differentiating between different days of the week or time of day, or different kinds of processes. Given a proper interface, such as can be provided by Excel (cf. Figure 1 below), the rules can be defined and maintained by business analysts who understand the actual thresholds for different resource types for different time periods. Thus, the rules can be easily customised for different (types of) data centres.

Table 1. A simple rule set

event	condition	action
ϵ	$\sigma_1 \leq 100$	set <i>label</i> = high
ϵ	$\sigma_1 \leq 90$	set <i>label</i> = medium
ϵ	$\sigma_1 \leq 60$	set <i>label</i> = low

Table 2. The previous rule set extended

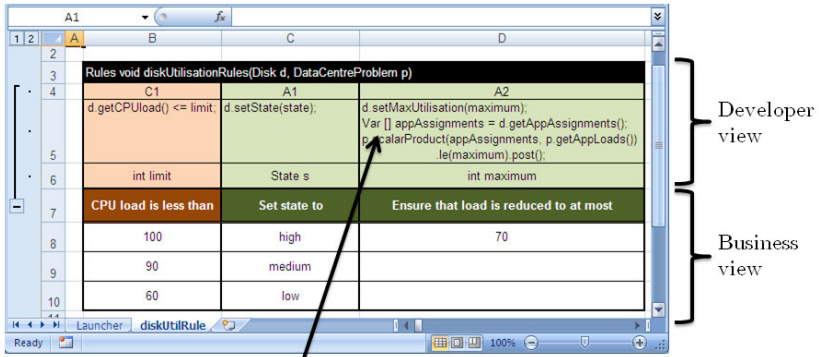
event	condition	action	additional constraints
ϵ	$\sigma_1(s) \leq 100$	set <i>label</i> = high	post $\sigma_1^{t+1}(s) \leq 70$
ϵ	$\sigma_1(s) \leq 90$	set <i>label</i> = medium	
ϵ	$\sigma_1(s) \leq 60$	set <i>label</i> = low	

3.2 Stage 1b: Building the Optimisation Model

At the problem identification stage we can also start to build the features of the optimisation model in terms of the variables and constraints. Our solution provides the facility to describe constraints within the Business Rules environment and to add them to a CP model. For example, we can extend the rules in Table 1 with constraints on the utilisation of a server over the next period. Again, let σ_1 denote the CPU load, and assume that we want to constrain the CPU load for servers with a high load to at most 70%. This can be reflected in the business rules by including a statement that adds (posts) a new constraint, as shown in Table 2. These same rules, but now specified in our Excel interface, are shown in Figure 11. The developer view (the lightly coloured rows 4, 5 and 6) is normally hidden from the user’s view, i.e. the business analyst would only see rows 7-10, and can manipulate the data there to reflect their situation (including the addition of rows to add more rules). The developer view shows how the rules relate to the business objects (e.g. specifying how the “CPU load is less than” condition is evaluated), and the effects on the business objects and the CSP. In this case, for example, the effect on the CSP is the creation of a new variable and the introduction of constraints on this variable, using the following Java snippet:

```
d.setMaxUtilisation(maximum);
Var [] appAssignments = d.getAppAssignments();
p.scalarProduct(appAssignments, p.getAppLoads()).le(maximum).post();
```

Besides the posting of simple constraints, we can introduce a certain amount of intelligence in the building of the model at this stage. As the number of servers can be very large, we believe that we should have the facility to limit the number of servers to consider in the solution. For example, we could limit ourselves to only those servers that have a high load, and those that have capacity to spare (e.g. those servers s for which $\sigma_1(s) \leq 30$). Again, the rules interface allows a data centre administrator to specify this efficiently.



Java snippet that creates appropriate variables and constraints

Fig. 1. The rule set of Table 2 specified in Excel

3.3 Stage 2: Solving the Optimisation Model

With the number of servers totaling potentially a very high number, the problem may quickly become intractable. Therefore, we propose that a smaller model can be generated, depending on the size and nature of the problem defined by the rules. Our hypothesis is that a small problem will find better solutions within the available time than a larger model, using the same search strategy.

To explore this hypothesis, we propose two models: a global one in which we take into account the full problem, and a localised one. In the first (full) model, we look at the state of all servers, identify which servers require a reallocation of volumes and solve the resulting problem.

The second approach is described by the following two inter-leaving stages:

1. As each server is sequentially identified as high risk using the rules definition, a small local model is created to solve the problem, but within a restricted set of servers that have spare capacity available; followed by
2. An optimisation stage which generates a new workload allocation with as few changes as possible.

This approach is particularly relevant where the presence of high risk servers is fairly sparse and there is the opportunity to create several small problems.

4 Experimental Results

4.1 Experimental Setup

Following discussions with a large data centre hardware provider, we generated a set of benchmark problems to test our system with. The benchmark set focuses on the management of storage servers. Such servers do not run any applications,

Table 3. Classification rules, $\sigma_1^1, \dots, \sigma_1^z$ are the last z values of the load sensors; I is the interval between readings. For brevity, the event column is omitted, as is the Java code generating the model.

condition	action
$\ \{\sigma_1^i \sigma_1^i > 45\}\ _z \geq 40$	set label = medium
$I \geq 10 \wedge \exists i \in 1 \dots z \cdot \sigma_1^i \geq 60$	set label = high
$I < 10 \wedge \exists i \in 1 \dots z \cdot \sigma_1^i \geq 70$	set label = high
$I \leq 15 \wedge \exists i \in 1 \dots z - 2 \cdot [\sigma_1^i \geq 70 \wedge \sigma_1^{i+1} \geq 70 \wedge \sigma_1^{i+2} \geq 70]$	set label = very high
$I \geq 15 \wedge \exists i \in 1 \dots z - 1 \cdot [\sigma_1^i \geq 70 \wedge \sigma_1^{i+1} \geq 70]$	set label = very high
$I \geq 30 \wedge \exists i \in 1 \dots z \cdot \sigma_1^i \geq 70$	set label = very high

yet are dedicated to the storage and retrieval of information. Information is stored on virtual disks (“volumes”) that may span many physical disks, even across different servers. Each volume has a list of applications that are using that particular volume, and a list of I/O operations per application. Different volumes may occupy the same physical disk. Thus, the total amount of data that is being transferred to/from a disk is determined by the activity of the different volumes on it. If several volumes with high transfer rates are stored on the same disk, the disk is overloaded and response times drop. In such a case, we want to redistribute the volumes over the active disks in such a way as to alleviate any bottle necks. Moving a volume is a costly operation, however, and so we want to minimise the number of moves over time.

Due to the nature of the problem, a number of additional constraints are present. These relate to the physical hardware that the systems employ. In particular, because of service level agreements that are in place, we can only move (part of) a volume from one physical disk to another, if these disks are of the same size, speed and type. Additionally, different parts of a RAID volume cannot share the same physical disk.⁴

Table 3 lists the rules that are used to classify the servers. Notice that the interval at which sensors are read differs between systems. Therefore, the time between sensor updates is included in the rules.

Description of the Benchmark Sets. Our benchmark set consists of 100 problem instances that were randomly generated from actual data. The problems have the following characteristics: we consider a storage array with n disks, each of which is represented by a server in our model. The array is used by $2n$ applications, where each application a_i requires space on between 1 and 3 disks (we say that the *virtual disk* for application a_i requires between 1 and 3 *volumes*). If application a_i requires, say, space on 2 disks, this gives rise to 2

⁴ A RAID configuration (Redundant Array of Inexpensive Disks) allows data to be divided and replicated among multiple hard drives to increase data reliability and/or increase performance. These benefits are lost when the parts of the array ends up on the same disk.

processes in our model, a_i^1 and a_i^2 , that we have to accommodate. For security reasons, these cannot be allocated on the same disk (as they are part of the same RAID device). Therefore, an *incompatible* constraint is imposed over such sets of processes. Finally, only 4 applications can use a disk at the same time due to space restrictions. To this end, we introduce a sensor σ_{size} that reports $\sigma_{size}^t(p) = 25$ for each time step t and each process p . For any given server s , we require $\sigma_{size}(s) \leq 100$ for all time points. To reiterate, the disks in the storage array are represented by the servers in our model; the volumes on those disks are represented by the processes running on the servers. We assume unit distance between all disks, in effect minimising the total number of changes we make to the allocation of applications to disks.

The values of the load sensors were chosen such that either $1/4^{th}$ or $1/8^{th}$ of the disks is considered to have a high or very high load, and the goal of the system is to achieve a maximum load of 60% for any given disk. As we specified the interval between sensor readings to be $I = 15$, this corresponds to ensuring that none of the disks will be classified as either high or very high when considering just the next time point. For each combination of the total number of disks and the number of overloaded ones, we generated 10 instances.

Models Considered. As indicated in Section 3.3, we consider two models. The first is the “full” model, in which the full set of disks is taken into account. The other model, referred to as “iterative” considers each disk in turn, and if the rules indicate that this disk is classified as high or very high, we create a small CSP to solve the reallocation problem for this disk. At first, this CSP is created using the overloaded disk and all unclassified disks (i.e. those with a low level of load). However, if we fail to find a solution for this CSP, we expand it to include all disks classified as medium as well. In this manner, we process all disks until they all have satisfactory load levels (or until we discover we cannot find a solution to the problem).

Our hypothesis is that the larger (i.e. full) models suffer from scalability issues (the full problem more so than the restricted one). The localised model is not expected to suffer from such scalability issues, but does lead to lower quality solutions: we expect to use more moves to rectify the situation.

Implementation. We used the ThinkSmart Technologies Intellify platform [4] to build our system. Intellify is a Java platform that allows us to integrate the OpenRules BRMS [5] as well as various CP solver implementations. The results we present here are obtained using the Constrainer [6] solver. We also have working versions using Choco [7] and an initial implementation of the forthcoming JSR-331 standard [8].

4.2 Results

The experimental data were generated using Java 1.6 on an Intel Core 2 Duo P8700, running at 2.53 GHz and with 4 GB of memory available (although none

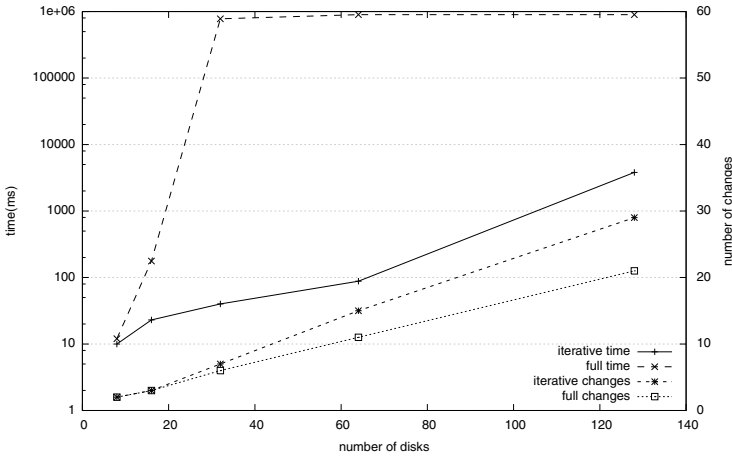


Fig. 2. Results with 12.5% of servers over loaded

of the experiments required more than 512 MB). We allowed each experiment a maximum of 15 minutes of CPU time (i.e. 900 seconds).

Figures 2 and 3 show the results of our experiments, taking the average of the 10 instances for each number of disks tested. Notice that the figures represent both the CPU time required to solve the instance, as well as the quality of the resulting solution (as measured in the number of moves). The two figures show a similar pattern: when the number of disks increases, solving the whole problem as one quickly leads to performance issues, and we cannot prove that we have found the optimal solution within 15 minutes. On the other hand, when we deal with each overloaded disk in isolation (and finding an optimal result during each iteration), it takes less than 10 seconds for the largest of problems. Thus, this part of our hypothesis is confirmed by these experiments. A χ^2 analysis shows that this is a statistically significant difference, as shown in Table 4.

The other part of our hypothesis stated that we expect to find that by solving iteratively, we produce lower quality solutions, i.e. we require more changes to achieve the desired load levels. Again, the figures confirm this hypothesis, as does the χ^2 analysis (cf. Table 4).

A closer examination of the results, shows exactly why this happens. When we look at the overloaded disks in isolation, the only way to deal with the issue of one of the disks being overloaded is to move at least 1 application away from the disk, and replace it with another volume (either belonging to some other application with a lighter load, or even an unused volume from another disk)⁵. Thus, 2 changes are required to deal with the issue. For example, consider 3 disks, d_1 , d_2 , and d_3 , each with two applications on them with loads: $d_1 = \{40, 25\}$,

⁵ Notice that the volumes in an array are of the same size. Disks are broken up into as many volumes of that size as possible (even if one or more of these will be unused) and thus no disks will have spare capacity to accommodate an additional volume. Hence, we always have to *swap* volumes.

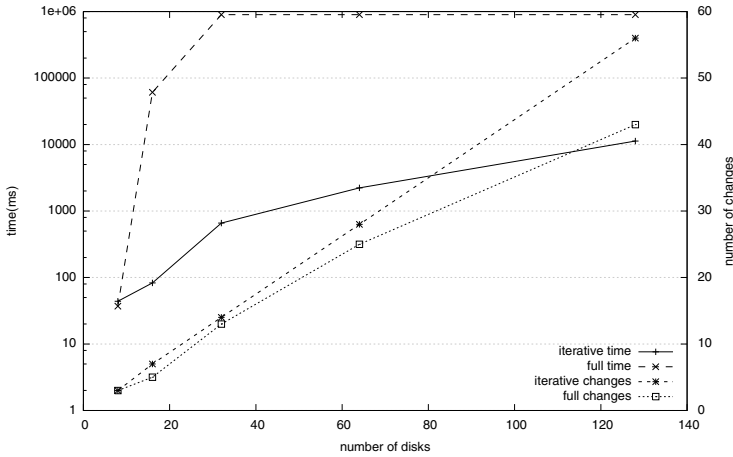


Fig. 3. Results with 25% of servers over loaded

$d_2 = \{40, 25\}$ and $d_3 = \{10, 10\}$. Suppose we want to achieve a maximum load of 60, and consider disk d_1 before disk d_2 . The local problem generated to deal with d_1 consists of disks d_1 and d_3 (d_2 is not considered, since it's overloaded). It is straight-forward to see that we can achieve the objective by swapping the application with load 25 with one of the applications from disk d_3 (totaling 2 moves). The same holds when we consider d_2 , and we achieve a total of 4 moves. Of course, in general, we may undo some of the moves we have made during an earlier iteration in order to satisfy the load on other disks, so we expect to see somewhere just under $2m$ changes given that there are m overloaded disks. (This number is confirmed by the graphs above.)

On the other hand, when we consider the solutions found when solving the whole problem at once, we observe that often, load is transferred between overloaded disks as well. In our previous example, for example, we can achieve a solution with only 3 moves: Move the application with load 40 from d_1 to d_3 , move the one with load 25 from d_2 to d_1 and move the one with load 10 from d_3 to d_2 . This requires that load is transferred between overloaded disks. Having this global view helps in detecting such situations, which explains the difference between the two approaches.

5 Related Work

There are a few instances of research in the combination of rules and constraints. Many of these works [9,10,11,12,13,14] relate to translating business rules into constraints and solving the problem with a single technology. This is not always applicable since business rules on their own can be used to make decisions using a different technology based on the RETE algorithm [15]. Perhaps closest to the research presented here is the work by Bousonville et al. [16], who use Business Rules to generate the data input to a predefined optimisation model.

Table 4. χ^2 analysis of the results, counting the number of problem instances one approach out-performs the other

	$1/4th$ overloaded		$1/sth$ overloaded	
	CPU time	Quality	CPU time	Quality
full model	4	41	7	33
iterative	46	0	43	0
χ^2	35.28	41	25.92	33
p	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$	$\ll 0.01$

In particular, they say, “we do not provide direct access to decision variables so that only predefined constraints are possible”. This is quite in contrast to what is demonstrated here. We indeed dynamically create constrained variables and post additional constraints from within the business rules environment. Rules are integral to solving the problem, not just as a pre-processor, but as an active component.

LAURE [17] (and its successor CLAIRE [18]) is a system which allows the use of rules to guide the optimisation search. *Constraint Handling Rules* (CHRs) are a more general approach to solving CSPs using rules. An overview of CHR approaches can be found in [19].

The automatic generation of dispatch rules (similar to business rules) for a packaging problem [20] through a CP optimisation model shows another hybrid configuration. However, there is a loose coupling between the technologies. Here an optimisation model, reflecting the global demands over a time period, is used to choose the parameters for a set of rules describing which task and which packaging machine to consider next.

Finally, related to our work are several approaches to *load balancing* in the data centre, such as, e.g. [21]. However, load balancing addresses the issue of how to distribute an incoming set of jobs (e.g. database queries or http requests) over the available servers, which is a different problem than the one we address here.

6 Conclusions and Future Work

In this paper, we describe a hybrid system that uses both a Business Rules Management System and a Constraint Programming solver to solve resource reallocation problems in large data centres. Such problems arise because the applications assigned to a server fluctuate in their requirements for, e.g. CPU power, memory, and bandwidth. When the demands on a server are larger than what it can provide, the performance degrades, and a solution has to be found through moving some of the workload to other servers.

In our solution, we use a BRMS to analyse the states of the servers over time. Due to stability reasons, we only want to move applications when a pattern has been established over several successive measurements, and the BRMS is used to

describe under which exact conditions servers are considered to be overloaded. Once the BRMS has established that one or more servers are overloaded, the rules that have fired are also used to construct a CSP corresponding to the problem. This is a distinguishing feature of our approach, as heretofore rules have only been used to setup parametrised models or to provide input to a CSP model. In contrast, our system allows the rules to create and access directly the variables and constraints in the CSP. This gives the rule administrator full control over how the problem is generated and solved, hence allowing the solution to be customised to the specific circumstances in their data centre.

The combination of Business Rules and CP is a powerful approach to the problem. For Business Rules, finding an optimal solution is problematic and could conceivably lead to many rules which are difficult to maintain. For Constraint Programming, the creation of the instance of the problem model would require much programming, and small changes to the underlying logic would require reprogramming that part of the solution. By passing each challenge to a different technology and linking them together at a low level delivers a fast, easily maintainable solution.

We have tested the system using data from a large data storage centre. The results show that while achieving optimal solutions is impractical for large problems (due to the time involved in constructing those optimal solutions), we are able to solve the problem within seconds by iteratively dealing with each overloaded resource in turn. However, this comes at a price, as the quality of the solution found is lower.

There are several areas for improvement to the current system. First of all, we want to examine the trade-off between solving iteratively and solving the problem as a whole. We identified specific conditions that improve the solution quality when solving the full problem (i.e. moving applications between overloaded disks in addition to the moving of applications between overloaded disks and those that are not overloaded that happens in the iterative solution strategy). By solving slightly larger problems, we may be able to achieve the best of both. This could even be set by a rule in the BRMS, allowing the data centre administrators to make the trade-off between solution quality and speed of achieving the solution.

Secondly, energy consumption is becoming a major issue for data centres. For this reason, we want to introduce the ability to recommend shutting down servers when not all servers are required, and *vice versa*, to recommend turning some additional servers on when there is a serious resource deficiency. This can be modeled already within the formal model of Section 2 (by introducing all-consuming dummy processes that are bound to servers when they are not available), but we have not explored this aspect using real data yet.

Acknowledgments

This work is supported by Enterprise Ireland under Grant PC/2009/0224 “Applying Integrated Rules and Constraint Technology to Diagnose and Resolve Capacity Deficiency Problems for Large Data Centres”, CP/2009/0201 and CFTD/06/209; and by Science Foundation Ireland under Grant 05/IN/1886.

References

1. Barroso, L., Hölzle, U.: The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture* 4, 1–108 (2009)
2. <http://www.vkernel.com/>
3. Herbst, H., Knolmayer, G., Myrach, T., Schlesinger, M.: The specification of business rules: A comparison of selected methodologies. In: *Tools for the Information System Life Cycle* (1994)
4. <http://www.thinksmarttechnologies.com/>
5. <http://www.openrules.com/>
6. <http://www.constrainer.sourceforge.net/>
7. <http://www.emn.fr/z-info/choco-solver/>
8. <http://4c110.ucc.ie/cpstandards/index.php/en/standards/java/jsr-331>
9. Kameshwaran, S., Narahari, Y., Rosa, C., Kulkarni, D., Tew, J.: Multiattribute electronic procurement using goal programming. *European Journal of Operational Research* 179(2), 518–536 (2007)
10. Carlsson, M., Beldiceanu, N., Martin, J.: A geometric constraint over k-dimensional objects and shapes subject to business rules. In: Stuckey, P.J. (ed.) *CP 2008*. LNCS, vol. 5202, pp. 220–234. Springer, Heidelberg (2008)
11. Fages, F., Martin, J.: From rules to constraint programs with the Rules2CP modelling language. In: *Recent Advances in Constraints* (2009)
12. Feldman, J., Korolov, A., Meshcheryakov, S., Shor, S.: Hybrid use of rule and constraint engines (patent no: WO/2003/001322) (2003)
13. Feldman, J., Freuder, E.: Integrating business rules and constraint programming technologies for EDM. In: *The 11th International Business Rules Forum and The First EDM Summit* (2008)
14. O’Sullivan, B., Feldman, J.: Using hard and soft rules to define and solve optimization problems. In: *The 12th International Business Rules Forum* (2009)
15. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* 19, 17–37 (1982)
16. Bousonville, T., Focacci, F., Pape, C.L., Nuijten, W., Paulin, F., Puget, J.F., Robert, A., Sadeghin, A.: Integration of rules and optimization in plant powerops. In: van Beek, P. (ed.) *CP 2005*. LNCS, vol. 3709, pp. 1–15. Springer, Heidelberg (2005)
17. Caseau, Y., Koppstein, P.: A cooperative-architecture expert system for solving large time/travel assignment problems. In: *Proceedings of the International Conference on Database and Expert Systems Applications*, pp. 197–202 (1992)
18. Caseau, Y., Laburthe, F.: CLAIRE: Combining objects and rules for problem solving. In: *Proceedings of the JICSLP 1996 Workshop on Multi-Paradigm Logic Programming* (1996)
19. Sneyers, J., van Weert, P., Schrijvers, T., de Koninck, L.: As time goes by: Constraint handling rules, a survey of chr research from 1998 to 2007. *Theory and Practice of Logic Programming* 10, 1–48 (2010)
20. van der Krogt, R., Little, J.: Optimising machine selection rules for sequence dependent setups with an application to cartoning. In: *Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing*, pp. 1148–1153 (2009)
21. Pinheiro, E., Bianchini, R., Carrera, E.V., Heath, T.: Load balancing and unbalancing for power and performance in cluster-based systems. In: *Workshop on Compilers and Operating Systems for Low Power* (2001)

Context-Sensitive Call Control Using Constraints and Rules

David Lesaint¹, Deepak Mehta², Barry O’Sullivan², Luis Quesada², and Nic Wilson²

¹ BT Research & Technology, BT, UK
dvdlst@gmail.com

² Cork Constraint Computation Centre, University College Cork, Ireland
{d.mehta,b.osullivan,l.quesada,n.wilson}@4c.ucc.ie

Abstract. Personalisation and context-awareness are fundamental concerns in Telephony. This paper introduces a rule-based system - 4CRULES - which enables context-sensitive call control by the means of feature configuration rules. 4CRULES is interoperable with standard context services and compositional feature architectures. It has been designed to resolve feature interactions, manage conflicting preferences, and mitigate the uncertainty affecting context data. This is achieved through a constraint optimisation model that maximises adherence to user requirements and domain constraints. Experiments on a suite of instances confirm the practicality of the approach and highlight performance- and adherence-critical factors.

1 Introduction

Telecommunications services like instant messaging or internet telephony bring increased flexibility to communicate at home, in the office or on the move. Their pervasiveness is also a source of disruptions and intrusions. Service providers are therefore looking for personalisation solutions allowing users to control the timing and modalities of their communications. In the case of telephony services, personalisation solutions are built around call control features. Technically, a feature is an increment of functionality that modifies the basic system behaviour. Dozens of call control features have been created to address concerns such as mobility, privacy, presentation, or billing. Features are optional and must be configured off-line to fulfil their role. Once configured, they execute by responding to call events (e.g., Call-Divert-On-Busy) and/or user actions (e.g., Call-Transfer) during calls.

Key requirements that drive the design of feature-rich telephony systems are: the ability for users to parametrise features (e.g., “Call-Divert to mobile”), address caller and callee scenarios (e.g., “Do-Not-Disturb and Speed-Dial”), combine or sequence features (e.g., “Call-Screen then Call-Divert”), request context-sensitive feature configurations (e.g., “Mute during seminars”), and express preferences or priorities (e.g., call policies imposed on a workforce). Different approaches ranging from scripting to policy enforcement have been proposed to meet these requirements. None, however, provides a comprehensive personalisation solution to: resolve undesirable feature interactions arising due to compositionality; manage conflicting preferences; and handle the uncertainty inherent to context data. To address these issues this paper presents a system

for Context-sensitive Configuration of Call Control features using Rules (4CRULES). 4CRULES allows a user to describe the behaviour of the communication service through a set of Feature Configuration Rules (FCRs). Conceptually, a FCR is weighted and associates a context condition to a feature subscription, which is defined to be a set of features, and a set of precedence constraints prescribed by a user and the feature catalogue. Each time a user's context is updated, the engine of 4CRULES infers a sequence of features from his/her set of FCRs. This sequence is free of undesirable feature interactions and it is applied to all calls involving the user until his context changes again.

A set of FCRs that are applicable for a given current context of a user could be inconsistent due to a variety of reasons as explained in the later sections. The 4CRULES engine computes a maximal subset of the applicable FCRs that is consistent and optimal in some sense. The optimality is based on a strict weak ordering over FCRs, which is obtained by evaluating a value for each FCR by combining priority of the FCR, concreteness of the context condition of the FCR, and probability of applicability of the FCR. The principle of the presented approach is to view FCRs processing as a constraint optimisation task. Overall, the method ensures maximum adherence to user requirements and interaction constraints. Experiments on a suite of instances confirm its practicality in terms of response time and highlight performance- and adherence-critical factors.

The next section provides background and reviews prior art on the call feature configuration. The architecture of 4CRULES and the principles of its implementation are then introduced. This is followed by a description of the FCRs language and processing method. The paper concludes with experiments using the existing implementation.

2 Related Work

From a user perspective, telephony systems that provide personalisation solutions must provide support for:

- **parameterisation:** some features use operational data supplied by users;
- **role-based selection:** some features apply to outgoing calls, some to incoming calls and others indistinctly (e.g., Call-Waiting);
- **composition:** features provide distinct functionalities and users need the flexibility to combine them and, in some cases, prioritise their execution;
- **contextualisation:** call handling requirements often depend on context, i.e., on extrinsic call characteristics such as user activity or location, and call service behaviour must be adapted accordingly;
- **preferences and priorities:** requirements may be weighted to enforce pre-emptive rights when call control is shared or to help resolve conflicts; and
- **uncertainty management:** information on context may be imprecise (e.g., coarse-grain localisation), incomplete (e.g., unclassified activity in a calendar service) or unavailable (e.g., off-line presence service).

Another critical requirement is the ability to manage feature interactions [1]. A feature interaction is “some way in which a feature modifies or influences the behaviour of another feature in generating the system's overall behaviour” [2]. Some interactions are

desirable and must be enabled to achieve proper system behaviour, e.g., when the modules of a feature must interact. Other interactions are undesirable and must be avoided. For instance, Call-Waiting-On-Busy and Call-Divert-On-Busy are triggered by the same event but take conflicting actions, i.e., put the caller on hold or divert his call.

Modern telephony systems, notably those based on the Session Initiation Protocol (SIP), rest upon compositional application architectures that are opened to personalisation and feature interaction management [3]. SIP is an application layer signalling protocol used to establish, modify and terminate multimedia sessions [4,5]. Various application programming interfaces (APIs) and domain-specific languages (DSLs) have been proposed to develop SIP services. These capabilities hide away low-level SIP stack operations to facilitate programmatic control over the call logic.

Scripting and policy languages are the main form of DSLs for SIP services. CPL [6] and LESS [7], for instance, provide primitive events and actions to specify fine-grained call control scripts. Scripts are uploaded to devices or application servers and interpreted by scripting engines at runtime. Policy enforcement systems allow for more declarativity and expressiveness. Policies are well suited to capture trigger-response patterns that commonly define feature behaviour. APPEL, for instance, supports an event-condition-action syntax to guard call primitives with call events subject to context conditions [8,9]. APPEL also supports conflict resolution policies [10]. Conflicts may occur when different policies are triggered and alternative actions may be taken to process a call. It is for the user to define which actions conflict and, if so, which resolution policy to apply (e.g., discard or replace actions).

These solutions mostly rely on prioritisation to handle conflicts. In addition, they provide limited support to capture constraints relating to application compositionality and they do not handle context uncertainty. Constraint-based systems offer an alternative. Distributed Feature Compositions (DFC), for instance, is an abstract network architecture designed to manage interactions through feature sequencing and exclusion constraints [11,12,13]. Constraints are elicited by analysing pairs of features and identifying those which are interaction-prone. This is achieved manually, formally or through simulation and testing [14,15,16,17].

Features and constraints are recorded in a catalogue and users configure their subscription by selecting and sequencing features accordingly. DFC routers then access user subscriptions to activate features sequentially when calls are set up. [18] propose a constraint optimisation approach to configure (context-agnostic) DFC feature subscriptions based on user preferences. 4CRULES builds upon this method to compute context-sensitive subscriptions.

3 Architecture and Principles of 4CRULES

4CRULES is a prototype designed to compute feature subscriptions of a user based on his/her real-time context information and feature configuration rules. It assumes a DFC-compliant telephony system that exposes a feature catalogue and accepts user subscriptions. It also assumes a context acquisition system that delivers user context information collected from different sources. Its logical architecture comprises a registrar, a rules-editor, a rules-store, a rules-engine, a context-log, and interfaces to the context acquisition and telephony systems - see Figure 1.

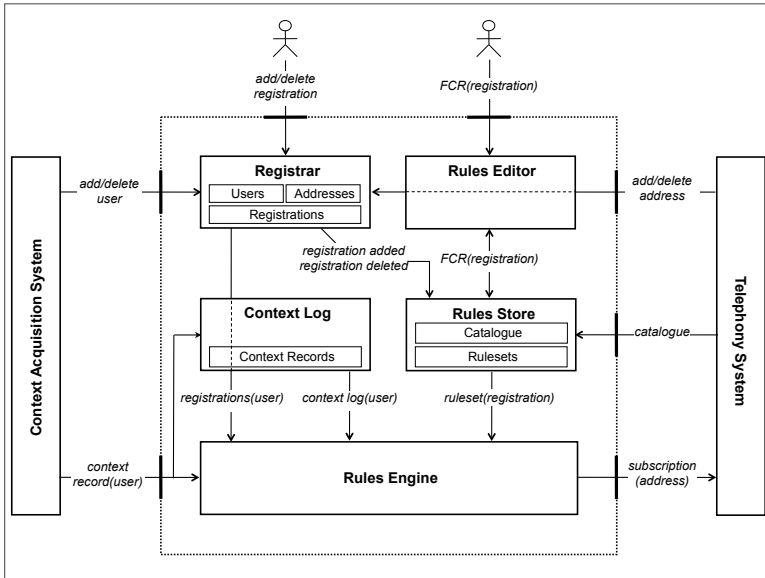


Fig. 1. The architecture of 4CRULES

The registrar associates users of the context acquisition system with addresses registered in the telephony system. The mapping between users and addresses is generally one-to-many. For simplicity, we identify users with addresses. 4CRULES associates a set of FCRs with each user. The rules-store maintains the rule-sets and the feature catalogue. The rules editor is the user interface to create, delete and update rule-sets.

The context acquisition system is responsible for tracking the context of users. It notifies the rules-engine when the context of a user changes and passes a record of the context of the user (context-record) which is also stored in the context-log. The engine retrieves the associated set of FCRs from the rules-store and computes a context-sensitive feature subscription. It then communicates the feature subscription to the telephony system. This feature subscription will prevail in all calls involving the user until a new context-record is received.

4CRULES shares a common meta-model with the context acquisition and telephony systems. The context meta-model is simple enough to achieve interoperability with a variety of services (e.g., GPS devices, calendar services, presence servers). It prescribes a finite domain representation for each context dimension (e.g., location) which is assumed to be exhaustive (i.e., no state omitted) and unambiguous (i.e., domain values denote distinct states). The granularity of context domains is unconstrained (e.g., days or quarters). Table 1 provides an example of a context model.

The (concrete) context of a user may thus be described with a single value from each domain of each dimension at any time. However, it is not always feasible to acquire concrete information about context dimensions. For this reason, 4CRULES accepts abstract contexts as input. Specifically, the context-records communicated by the

acquisition system may include alternative values for each dimension to denote multiple states. For instance, the abstract context “Friday, PM, lunch or visit” denoted $\langle D: \{\text{Friday}\}, H: \{\text{PM}\}, A: \{\text{lunch}, \text{visit}\} \rangle$ ¹ is a valid record in reference to the model of Table 1. Notice that no assumption is made about the frequency of communications. It is for the acquisition system to decide which record to communicate and when.

Table 1. A context model

Dimension	DAY	HOURS	ACTIVITY	LOCATION	PRESENCE
Domain	Monday	AM	journey	home	appearOffline
	Tuesday	PM	lunch	office	away
	Wednesday		standBy	anyOther	beRightBack
	Thursday		visit		busy
	Friday		anyOther		doNotDisturb
	Saturday				offline
	Sunday				online

As far as the telephony system is concerned, the meta-models for feature catalogue and feature subscription subsume that of DFC. A catalogue is a set of features and precedence constraints, and the induced relation may be cyclic (i.e., some features may be incompatible) and non-transitive. A feature subscription is a set of features, user-defined precedence constraints, and a set of catalogue precedence constraints defined on the selected features. It is consistent if the induced relation is acyclic. Catalogue and subscription meta-models also accommodate parameter signatures for features.

The feature configuration rule language is built upon the context and the feature catalogue meta-models. The antecedent of a FCR specifies an abstract context which has Cartesian product semantics similarly to that of context-records. The consequent of a FCR is a feature subscription augmented with feature exclusion constraints. Basically, the user can express inclusion and exclusion constraints over individual features and precedence constraints over the included features. The priority of a FCR is selected by the user from a total order (e.g., $\text{low} < \text{medium} < \text{high}$). The region of a FCR is *target* (resp., *source*) if the user is callee (resp., caller).

Figure 2 illustrates a set of FCRs. The identifier of the first FCR is 1, its priority is high, its region *target*, and it prescribes the activation of feature *divert* with parameter value *addr1* if the activity recorded for the user is *journey* or *visit*. This means that all incoming calls received during journeys or visits should be forwarded to address *addr1*. The second rule prescribes the activation of feature *tScreen* before feature *divert* as indicated by keyword *BEFORE*. The third rule excludes feature *divert* as indicated by keyword *DONT*. In other words, this rule cannot be composed with a rule that requires *divert*. The next rule restricts the two dimensions *Day* and *Hour*. The last rule handles outgoing calls and is labelled with *source* instead of *target*.

Given an abstract context recorded for a user, and a set of FCRs provided by the user the rules-engine proceeds in two steps. It first identifies the set of FCRs that are applicable, i.e., whose antecedents intersect with the recorded abstract context. Notice that an applicable FCR does not necessarily subsume the concrete context of the user. For this

¹ We represent an abstract context by abbreviating dimensions with their initials (e.g., *D* for *DAY*) and omitting those that are not restricted (*LOCATION* and *PRESENCE* in this example).

[1, high, target] A: {journey, visit}	→ divert(addr1)
[2, high, target] L: {home}	→ tScreen(list1) BEFORE divert(addr1)
[3, low, target] A: {lunch}	→ play(away.mp3) DONT(divert)
[4, low, target] D: {Monday, Friday} AND H: {PM}	→ divertNoAnswer(addr2)
[5, low, target] L: {anyOther}	→ play(welcome.mp3)
[6, high, source] L: {anyOther}	→ oScreen(list2)

Fig. 2. A set of feature configuration rules

reason, the engine computes a probability of applicability for each FCR as described in Section 6. The second step computes an interaction-free sequence of features that is obtained by composing the consequents of applicable FCRs. Since applicable rules may not be consistent as described in Section 5, the engine determines a consistent relaxation, i.e., a subset of rules that are consistent. Since there may be many such relaxations, the engine computes an optimal relaxation using a lexicographic order based on rule priority, concreteness and probability of applicability. The engine solves this combinatorial optimisation problem using a constraint programming model.

The next sections present the notations and definitions related to the context and catalogue meta-models, the rules language, the notions of probability of applicability and relaxation, and the constraint programming formulation of the relaxation problem.

4 Context and Catalogue Meta-models

In this section we describe the context and catalogue meta-models.

4.1 Context Dimensions and Records

A *context model* is a tuple of context dimensions. A *context dimension* is represented by a finite domain of values called *context domain*. Let \mathcal{M} denote a context model, m the number of context dimensions of \mathcal{M} , D_i , $1 \leq i \leq m$, the domain associated with the i^{th} dimension of \mathcal{M} , and d the size of the largest domain in \mathcal{M} . Without loss of generality we assume that each context domain is totally ordered.

An *abstract context* $a = \langle a_1, a_2, \dots, a_m \rangle$ over \mathcal{M} is a tuple consisting of a (non-empty) set of values per context domain. If all sets in a are singleton, a is said to be *concrete*. For instance, $\langle D: \{\text{Monday, Tuesday}\}, A: \{\text{lunch, visit}\} \rangle$ is an abstract context while $\langle D: \{\text{Monday}\}, H: \{\text{PM}\}, A: \{\text{lunch}\}, L: \{\text{office}\}, P: \{\text{offline}\} \rangle$ is a concrete context wrt. Figure 1. Whenever a dimension is not specified (e.g., L in the previous abstract context) the full domain is assumed. A *context-record* is an abstract context over \mathcal{M} .

Let $a = \langle a_1, a_2, \dots, a_m \rangle$ be an abstract context over \mathcal{M} . $\llbracket a \rrbracket$ denotes the Cartesian product $a_1 \times a_2 \times \dots \times a_m$. Let $a_i = \langle a_{i_1}, \dots, a_{i_m} \rangle$ and $a_j = \langle a_{j_1}, \dots, a_{j_m} \rangle$ be two abstract contexts over \mathcal{M} . We say that a_i *subsumes* a_j , denoted by $a_j \subseteq a_i$ if and only if $(a_{j_1} \subseteq a_{i_1}) \wedge \dots \wedge (a_{j_m} \subseteq a_{i_m})$. The *intersection* of a_i and a_j , denoted by $a_i \cap a_j$, is $\langle (a_{i_1} \cap a_{j_1}), \dots, (a_{i_m} \cap a_{j_m}) \rangle$. The complexity of context intersection is linear in the number of dimensions and linear in the maximum domain size. The *maximal abstract context* over \mathcal{M} for subsumption is denoted D , i.e., $D = \langle D_1, \dots, D_m \rangle$.

4.2 Feature Catalogues and Subscriptions

A *feature catalogue* is a pair $\langle \mathcal{F}, \mathcal{H} \rangle$, where \mathcal{F} is a set of features and \mathcal{H} is a set of hard precedence constraints on \mathcal{F} . A precedence constraint, $i \prec j \in \mathcal{H}$, means that if the features i and j are part of a subscription then i must precede j in that subscription. Two features i and j are mutually exclusive if they can never appear together in any subscription. This is expressed by a pair of precedence constraints $i \prec j$ and $j \prec i$.

The name of a feature together with the name and type of its parameters define the *signature* of a feature. For instance, feature `play` in Figure 2 is parameterised with the name of a media file whereas `tScreen` is parameterised with a list of addresses. For each feature $f \in \mathcal{F}$, s_f denotes the signature of f . S denotes the set of feature signatures, i.e., $S = \{s_f | f \in \mathcal{F}\}$, and s the largest number of parameters for a feature.

Formally, a feature subscription for a catalogue $\langle \mathcal{F}, \mathcal{H} \rangle$ is a tuple $\langle F, V, H \cup P \rangle$, where $F \subseteq \mathcal{F}$, V is a set of feature parameter assignments complying with signature $s_f \in S$ for each $f \in F$, H is the projection of \mathcal{H} on F , i.e., $\mathcal{H} \downarrow_F = \{(i \prec j) \in \mathcal{H} : i, j \in F\}$, and P is a set of user-defined precedence constraints on F . A feature subscription $\langle F, V, H \cup P \rangle$ is defined to be *consistent* if and only if the directed graph $\langle F, H \cup P \rangle$ is acyclic. For instance, a subscription $\langle \{\text{tScreen}, \text{divert}\}, \{\text{tScreen:} \text{list} = \text{11}, \text{divert:} \text{addr} = \text{a1}\}, \{\text{tScreen} \prec \text{divert}\} \rangle$ is acyclic and hence it is consistent.

Let $U_1 = \langle F_1, V_1, H_1 \cup P_1 \rangle$ and $U_2 = \langle F_2, V_2, H_2 \cup P_2 \rangle$ be two feature subscriptions. Let $v_{f_i} \in V_i$ denote the parameter assignment for $f \in F_i$ in U_i , $1 \leq i \leq 2$. We say that U_1 and U_2 are *unifiable* if and only if for all $f \in F_1 \cap F_2$, $v_{f_1} = v_{f_2}$. For instance, $\langle \{\text{divert}\}, \{\text{divert:} \text{addr} = \text{a1}\}, \emptyset \rangle$ and $\langle \{\text{tScreen}, \text{divert}\}, \{\text{tScreen:} \text{list} = \text{11}, \text{divert:} \text{addr} = \text{a1}\}, \{\text{tScreen} \prec \text{divert}\} \rangle$ are unifiable subscriptions since feature `divert` has the same parameter assignment in each subscription. The *composition* of n subscriptions $\langle F_i, V_i, H_i \cup P_i \rangle$, $1 \leq i \leq n$, is the subscription $\langle F_c, V_c, H_c \cup P_c \rangle$, where $F_c = F_1 \cup \dots \cup F_n$, $V_c = V_1 \cup \dots \cup V_n$, $H_c = \mathcal{H} \downarrow_{F_c}$, and $P_c = P_1 \cup \dots \cup P_n$. For instance, the composition of the two subscriptions $\langle \{\text{divert}\}, \{\text{divert:} \text{addr} = \text{a1}\}, \emptyset \rangle$ and $\langle \{\text{divertNoAnswer}\}, \{\text{divertNoAnswer:} \text{addr} = \text{a2}\}, \emptyset \rangle$ is the subscription $\langle \{\text{divert}, \text{divertNoAnswer}\}, \{\text{divert:} \text{addr} = \text{a1}, \text{divertNoAnswer:} \text{addr} = \text{a2}\}, \{\text{divertNoAnswer} \prec \text{divert}, \text{divert} \prec \text{divertNoAnswer}\} \rangle$ if we assume that `divert` and `divertNoAnswer` are mutually exclusive in the catalogue.

5 Feature Configuration Rules

A feature configuration rule (FCR) associates an abstract context to a consistent feature subscription. Let r be a FCR. $a_r = \langle a_{r_1}, \dots, a_{r_m} \rangle$ denotes the abstract context of r where $\forall i, 1 \leq i \leq m$, $a_{r_i} \neq \emptyset$. $s_r = \langle F_r, V_r, H_r \cup P_r \rangle$ denotes the consistent feature subscription of r . We say that a FCR r is *applicable* to a given context-record ς if and only if a_r and ς have a common intersection, i.e., $\llbracket a_r \cap \varsigma \rrbracket \neq \emptyset$. For instance, FCR 1 in Figure 2 is applicable to $\langle \text{A:} \{\text{journey}, \text{standBy}\} \rangle$ but FCR 3 is not. Let R be a set of FCRs. We say that R is applicable to an abstract context ς if and only if ς intersects with each antecedent, i.e., $\forall r \in R$, $\llbracket a_r \cap \varsigma \rrbracket \neq \emptyset$. We say that the FCRs in R are *mutually applicable* to ς if and only if their antecedents and ς have a common intersection, i.e., $\llbracket \bigcap_{r \in R} a_r \cap \varsigma \rrbracket \neq \emptyset$. For instance, the FCRs of Figure 2

are individually applicable but they are not mutually applicable to the abstract context $\langle A: \{journey, lunch\}, L: \{home, anyOther\} \rangle$.

We want to allow the possibility of expressing that some feature cannot be part of the final subscription should the FCR be applied and composed with other FCRs. In order to allow that, for each feature $f \in \mathcal{F}$, a dummy feature \bar{f} and two precedence constraints $f \prec \bar{f}$ and $f \succ \bar{f}$ are added to the catalogue to enforce mutual exclusion. The dummy feature \bar{f} can then be included in the subscription to specify the exclusion of f should the rule be applied. This ensures that a FCR excluding f will be incompatible with a rule requesting f . We say that R is *compatible* with the catalogue if and only if the composition of the subscriptions prescribed by the rules of R denoted $\langle F_R, V_R, H_R \cup P_R \rangle$ is consistent. For instance, the rule-set $\{1, 2\}$ is *compatible* but $\{1, 3\}$ is not due to the mutual exclusion constraint between `divert` and `divert` in the catalogue. In the following we shall also assume that features `divert` and `divertNoAnswer` are mutually exclusive in the catalogue and that no other catalogue constraint applies to the features used in the rule-set in Figure 2. Therefore, the rule-set $\{1, 4\}$ is not compatible with the catalogue.

We say that R is *unifiable* if and only if the feature subscriptions prescribed by FCRs of R are pairwise unifiable. For instance, the rule-set in Figure 2 is not unifiable due to the assignment of feature `play` in rule 3 and 5. Figure 3 shows the pairs of rules in Figure 2 that are not compatible, not unifiable or not mutually applicable to the abstract context $\varsigma_1 = \langle D: \{Friday\}, H: \{PM\}, A: \{journey, lunch\}, L: \{home, anyOther\}, P: \{office\} \rangle$.

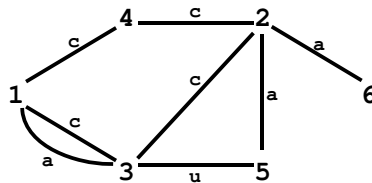


Fig. 3. Graph of inconsistent pairs of rules in Table 2 wrt. abstract context ς_1 . Nodes represent rules and edges labelled with c (respectively, u, a) connect rules that are not compatible (resp., unifiable, mutually applicable to ς_1).

Given a set of FCRs R which is applicable to an abstract context ς , we say that R is *consistent* with ς if and only if the FCRs of R are mutually applicable to ς , R is unifiable, and R is compatible with the catalogue. For instance, the set of rules $\{3, 4, 6\}$ in Figure 2 is consistent with the abstract context ς_1 and corresponds to one of the independent sets in the inconsistency graph shown in Figure 3. Note that independence in the inconsistency graph is necessary but insufficient in the general case to ensure consistency since the rules of an inconsistent set of rules may be pairwise consistent. Each FCR r is associated with a weight that includes a user-defined priority, a concreteness measure, and a probability of applicability. The concreteness of a rule is fixed and represents the cardinality of its abstract context whereas its probability of applicability is relative to the context-record being considered.

Proposition 1. *Let R be a set of FCRs applicable to an abstract context ς . The time complexity of checking the consistency of R with ς is $\mathcal{O}(|R|(|F_R|s + md) + |H_R \cup P_R|)$.*

Proof. The time complexity of checking the applicability of R is $\mathcal{O}(|R|md)$, where m is the number of context dimensions and d is the maximum domain size of the context dimensions. This is because the time-complexity of computing the intersection of two abstract contexts is $\mathcal{O}(md)$ and $|R|$ such intersections are required. The complexity of checking the compatibility of R is equivalent to checking the consistency of the feature subscription $\langle F_R, V_R, P_R \cup H_R \rangle$, which is $\mathcal{O}(|F_R| + |P_R \cup H_R|)$. The complexity of checking the unifiability of FCRs is $\mathcal{O}(|F_R||R|s)$. This is because verifying whether a feature has the set of parameter values in different rules is $\mathcal{O}(|R|s)$ and checking this for all features is $\mathcal{O}(|F_R||R|s)$. Thus, the overall time complexity is $\mathcal{O}(|R|(|F_R|s + md) + |H_R \cup P_R|)$.

6 Probability of Applicability of a FCR

4CRULES assumes that any context-record, ς , sent by the context acquisition system for a user includes his/her current context, i.e., the concrete context denoting his state. Consistently with this assumption, the rules-engine discards rules that are not applicable, i.e, rules whose antecedents do not intersect with ς . Since their antecedents cannot include the current context, it is safe not to apply them. The method, however, considers any other rule as applicable. While this is safe for rules whose antecedents subsume the context-record, and therefore includes the current context, this is not necessarily safe for those rules whose antecedents do not subsume the context-record. For such rules, it cannot be exactly ascertained whether the antecedent includes the current context or not.

For this reason, the rules-engine computes a probability of applicability for each rule with respect to the context-record. This probability is based on a probability distribution over the space of concrete contexts that is specific to the user. Given a user u , the associated probability distribution, a context-record ς , and the antecedent a_r of a rule r associated with u , the probability that r is applicable to ς is the sum of the probabilities of the concrete contexts that are common to a_r and ς . By default, the distribution may be assumed uniform, i.e., each concrete context is equally likely to occur. In this case, the probability of applicability of a rule r to a context-record ς , denoted $Pu(a_r|\varsigma)$, is defined by $Pu(a_r|\varsigma) = \frac{|\llbracket a_r \wedge \varsigma \rrbracket|}{|\llbracket \varsigma \rrbracket|}$. For example, if $\varsigma = \langle L: \{\text{anyOther}, \text{office}\}, A: \{\text{lunch}\} \rangle$, then $Pu(a_6|\varsigma) = 1/2$. Here a_6 denotes the abstract context $\langle L: \{\text{anyOther}\} \rangle$ associated with FCR 6 in Figure 2. Notice that $a_6 \wedge \varsigma$ and ς differ only in the dimension Location: the former has only one value for the dimension Location while the latter has two values.

Alternatively, a frequency distribution may be used if the context-records produced by the acquisition system are logged. The context-log of a user is a list of pairs $\langle a, f(a) \rangle$ where a is an abstract context and $f(a)$ its frequency. The log is initialised with the pair $\langle D, 1 \rangle$ corresponding to the maximal abstract context D . When the acquisition system produces an abstract context a for the user, his/her context-log is extended with a new pair $\langle a, 1 \rangle$ if there is no pair whose first element is a or else the frequency of a is

Algorithm 1. MCS($\mathcal{L}, \varsigma, R, k$)

Require: \mathcal{L} : context-log of a given user, ς : context-record of the user, R : set of FCRs of the user applicable to context ς , and k : number of trials.

Ensure: Pf_r is an approximation of $Pf(a_r|\varsigma)$.

```

1: for  $1 \leq r \leq |R|$  do
2:    $s_r \leftarrow 0, t_r \leftarrow 0$ 
3: for  $1 \leq q \leq k$  do
4:   randomly select an abstract context  $a$  from  $\mathcal{L}$  such that  $a \wedge \varsigma \neq \emptyset$  favouring those contexts
   with higher frequency.
5:   for all  $r = 1 \dots |R|$  do
6:      $s_r \leftarrow s_r + \llbracket a \wedge \varsigma \wedge a_r \rrbracket$ 
7:      $t_r \leftarrow t_r + \llbracket a \wedge \varsigma \rrbracket$ 
8: for  $r = 1 \dots |R|$  do
9:    $Pf_r \leftarrow s_r / t_r$ 

```

incremented by 1 in the existing pair $\langle a, f(a) \rangle$. Given a log of context-records \mathcal{L} , the probability of applicability of a rule r to a context-record ς , denoted $Pf(a_r|\varsigma)$, is

$$Pf(a_r|\varsigma) = \frac{\sum_{\langle a, f(a) \rangle \in \mathcal{L}} \llbracket a \wedge \varsigma \wedge a_r \rrbracket \times f(a)}{\sum_{\langle a, f(a) \rangle \in \mathcal{L}} \llbracket a \wedge \varsigma \rrbracket \times f(a)} \quad (1)$$

The denominator is the total count of the concrete contexts covered by the abstract contexts subsumed by ς according to \mathcal{L} , and the numerator is the number of times a user has been in one of the concrete contexts subsumed both by ς and a_r . Let $size(\mathcal{L}) = \sum_{a \in \mathcal{L}} f(a)$ be the *size* of a log \mathcal{L} . The exact computation of $Pf(a_r|\varsigma)$ is linear with respect to $size(\mathcal{L})$. If $size(\mathcal{L})$ is large, a Monte-Carlo method such as Algorithm 1 can be used for generating a close approximation. Given a context-log \mathcal{L} for a user u , a context-record ς , a set of rules R applicable to ς , and an integer k , Algorithm 1 randomly selects k abstract contexts from the log favouring those contexts with higher frequency and it updates two counters s_r and t_r for each rule r . Here t_r denotes the total count of the concrete contexts covered by k abstract contexts selected from the log that are subsumed by ς , and s_r denotes the portion of t_r that is consistent with a_r . The algorithm returns the ratio of s_r and t_r which approximates $Pf(a_r|\varsigma)$.

When $size(\mathcal{L})$ is small, it is reasonable to use $Pu(a_r|\varsigma)$ for computing the probability of applicability. As $size(\mathcal{L})$ increases, it is desirable to gradually switch to $pf(a_r|\varsigma)$. In order to achieve that, a weighted average of $Pu(a_r|\varsigma)$ and $Pf(a_r|\varsigma)$, denoted $Pa(a_r|\varsigma)$, may be used.

7 Optimal Relaxation of a Set of FCRs

Given a context-record ς and a set of FCRs \mathcal{R} , the engine searches for a consistent set of rules amongst \mathcal{R} . Let $R \subseteq \mathcal{R}$ be the set of rules applicable to ς . If the rules in R are mutually not applicable to ς , or R is non-unifiable, or it is incompatible with the catalogue then R is inconsistent, in which case the task is to find a relaxation of \mathcal{R} that is consistent. From now on, by relaxation we mean consistent relaxation. A consistent

Algorithm 2. $\lambda(r_i, r_j) : \text{Boolean}$

- 1: **If** $Pa_\epsilon(a_{r_i}/\zeta) > Pa_\epsilon(a_{r_j}/\zeta)$ **then return true**
 - 2: **else if** $Pa_\epsilon(a_{r_i}/\zeta) < Pa_\epsilon(a_{r_j}/\zeta)$ **then return false**
 - 3: **else if** $\kappa(r_i) > \kappa(r_j)$ **then return true**
 - 4: **else if** $\kappa(r_i) < \kappa(r_j)$ **then return false**
 - 5: **else if** $\gamma(r_i) < \gamma(r_j)$ **then return true**
 - 6: **else return false**
-

relaxation R' of R is *maximal* if there does not exist any relaxation R'' verifying $R' \subset R''$. Since there may be many maximal relaxations, the engine searches for a maximal relaxation that is optimal in some sense. The notion of optimality is defined using a lexicographic order over sets of FCRs. A lexicographic order over maximal relaxations can be derived from a strict weak ordering over rules.

A strict weak ordering is a binary relation on a set that is a strict partial order in which the relation “neither a is related to b nor b is related to a ” is transitive. Given a rule $r_i \in R$, $Pa(a_i|\zeta)$ denotes the probability that the rule is applicable to ζ , $\kappa(r_i)$ denotes the priority of the rule, and $\gamma(r_i)$ denotes the concreteness of the rule. Algorithm 2 introduces a strict weak ordering λ which compares rules based on their probability of applicability, priority, and concreteness. Given two rules, λ first compares their approximate probabilities of applicability. In the algorithm, $Pa_\epsilon(a_{r_i}/\zeta)$ means $Pa(a_{r_i}/\zeta)$ rounded to the nearest ϵ . For example, if ϵ is set to being 0.1 and $Pa(a_{r_i}/\zeta) = 0.53$ then $Pa_\epsilon(a_{r_i}/\zeta) = 0.5$. If the probabilities are approximately equal then λ compares their priorities. If the priorities are equal then λ compares their concreteness. If their concreteness are also identical then the two rules are λ -incomparable, i.e., equally important with respect to λ . An alternative strict weak ordering can be obtained by combining priority and probability of applicability, i.e., r_i is preferred to r_j if and only if $Pa(a_{r_i}/\zeta) \times \kappa(r_i) > Pa(a_{r_j}/\zeta) \times \kappa(r_j)$.

Given a strict weak ordering λ , $\Theta_\lambda(R)$ denotes the total ordering of R obtained by ordering its elements using λ and breaking the ties between λ -incomparable rules using their identifiers. Let R' and R'' be two consistent sets of rules such that $\Theta_\lambda(R') = \langle r'_1, \dots, r'_p \rangle$ and $\Theta_\lambda(R'') = \langle r''_1, \dots, r''_q \rangle$. R' is *lexicographically better* than R'' for λ , denoted $\Theta_\lambda(R') \prec_{lex} \Theta_\lambda(R'')$ if and only if

- $p > q$ and for all $i, i \leq q$, neither $\lambda(r'_i, r''_i)$ nor $\lambda(r''_i, r'_i)$ holds, or
- there exists $i \leq \min(p, q)$ such that $\lambda(r'_i, r''_i)$ holds and for all $l < i$, neither $\lambda(r'_l, r''_l)$ nor $\lambda(r''_l, r'_l)$ holds.

In words, R' is better than R'' if there exists $r'_i \in \Theta_\lambda(R')$ and $r''_i \in \Theta_\lambda(R'')$ such that r'_i is more important than r''_i with respect to λ and for all r'_l and r''_l occurring before r'_i and r''_i , r'_l and r''_l are equally important. This assumes that R' and R'' have the same number of rules. If not, the smaller one is padded with “blank rules”, where a blank rule is treated as a least important rule with respect to λ .

A relaxation R' of R is *optimal* if and only if there does not exist any relaxation R'' of R such that $\Theta_\lambda(R'') \prec_{lex} \Theta_\lambda(R')$. By definition of \prec_{lex} , an optimal relaxation is necessarily maximal with respect to set inclusion. If λ generates a total order on R then \prec_{lex} is itself a total order and finding an optimal relaxation of R is polynomial.

However, if λ is a strict weak order on R , then \prec_{lex} is also a strict weak ordering and finding an optimal relaxation of R is NP-hard. This can be proven by reducing the maximum independent set problem to the problem of finding a lexicographically optimal relaxation of a set of rules. Given a graph, a node can be associated with a rule and an edge between two nodes can be associated with an incompatibility between two rules. As finding a maximum independent set is NP-hard, finding a lexicographically optimal relaxation is also NP-hard. Notice that two rules can be incompatible: (1) when they are mutually not applicable, i.e., when their abstract contexts do not intersect, (2) when they are non-unifiable, i.e., when a common feature is assigned different parameter values, or (3) when the composition of their configurations is inconsistent.

Let us consider an example to demonstrate the concepts of maximal relaxation and optimal relaxation. Let $\varsigma_1 = \langle D: \{\text{Friday}\}, H: \{\text{PM}\}, A: \{\text{journey, lunch}\}, L: \{\text{home, anyOther}\}, P: \{\text{office}\} \rangle$ be an abstract context. All the rules in Figure 2 are applicable, since the intersection of ς_1 with each of them is non-empty. The probability of applicability of FCR 4 is 100%, while for others is 50%. The concreteness of FCR 4 is 210, that of FCR 3 is 294, that of FCRs 2, 5 and 6 is 490 and that of FCR 1 is 588. The strict weak ordering on the rule-set with respect to λ is $4 \prec \{2, 6\} \prec 1 \prec 3 \prec 5$. This rule-set is not consistent with ς_1 as shown in Figure 3. It has 4 maximal relaxations which are $A = \{2, 1\}$, $B = \{1, 6, 5\}$, $C = \{4, 6, 3\}$ and $D = \{4, 6, 5\}$. The lexicographic ordering over maximal relaxations with respect to \prec_{lex} is a strict total order equal to $C \prec D \prec B \prec A$. C is therefore the optimal relaxation.

8 A Constraint Optimisation Formulation

This section formulates the problem of finding a lexicographically optimal relaxation of an inconsistent set of FCRs as a constraint optimisation problem (COP). The COP is defined in terms of finite domain variables, constraints restricting the assignments of values to the variables, and an objective function. The variables model the inclusion of rules in the computed relaxation, the inclusion and positioning of features in the computed subscription, and the satisfaction of user precedences. The constraints model all the requirements—mutual applicability, unifiability and compatibility of the rule-set—whereas the objective function models the lexicographic order over rule-sets.

Variables and Domains. A Boolean variable br_i is associated with each rule $r_i \in R$, which is instantiated to 1 or 0 depending on whether r_i is included in the computed relaxation or not, respectively. Each $f_i \in F_R$ is associated with two variables: a *Boolean variable* bf_i and an *integer variable* pf_i . A variable bf_i is instantiated to 1 or 0 depending on whether f_i is included in the computed subscription or not, respectively. The domain of each variable pf_i is $\{1, \dots, |F_R|\}$. If f_i is included then pf_i denotes the position of f_i in a sequence. Each user precedence constraint $p_{ij} \equiv (f_i \prec f_j) \in P_R$ is associated with a *Boolean variable* bp_{ij} , which is instantiated to 1 or 0 depending on whether p_{ij} is respected in the computed subscription or not, respectively.

Constraints. A catalogue precedence constraint $(i \prec j) \in H_R$ can be expressed as $bf_i \wedge bf_j \Rightarrow (pf_i < pf_j)$, which is trivially satisfied when either bf_i or bf_j is

instantiated to 0. A user precedence constraint $(i \prec j) \in P_R$ can be expressed as $bp_{ij} \Rightarrow (bf_i \wedge bf_j \wedge (pf_i < pf_j))$. If it holds then f_i and f_j are included in the subscription and f_i is placed before f_j in the sequence. For each $f \in F_R$, if $f \in (F_{r_i} \cap F_{r_j})$ and $v_{f_i} \neq v_{f_j}$, then the rules r_i and r_j are non-unifiable. In order to ensure unifiability the constraint $\neg br_i \vee \neg br_j$ is added. The computed relaxation is a mutually applicable set of rules which is expressed by $\llbracket \bigwedge_{r_i \in R} br_i = 1 \ a_i \wedge \varsigma \rrbracket \neq \emptyset$. If r_i is included in the computed relaxation then its subscription is included in the computed subscription (i.e., the subscription induced by the computed relaxation). This is expressed by the constraint $br_i \Rightarrow \bigwedge_{f_j \in F_i} bf_j \wedge \bigwedge_{p_{kl} \in P_i} bp_{kl}$.

A feature f_j (respectively, a user precedence p_{kl}) can only be included in the computed subscription if it belongs to the subscription of a rule that is included in the computed subscription. For each feature $f_j \in F_R$ and each user precedence $p_{kl} \in P_R$, we add the constraints $bf_j \Rightarrow \bigvee_{(f_j \in F_i) \wedge (r_i \in R)} br_i$ and $bp_{kl} \Rightarrow \bigvee_{(p_{kl} \in P_i) \wedge (r_i \in R)} br_i$.

Objective Function. A solution of the COP model is a subscription induced from the set of the rules that are included. Let I be a subset of rules of R that are included, i.e., $\{r_i | r_i \in R \wedge br_i = 1\}$. The value of the solution is $\Theta_\lambda(I)$, which is an ordered set of rules based on the comparator λ . The objective is to find an ordered set of rules, $\Theta_\lambda(I)$, that is consistent and lexicographically optimal.

9 Empirical Evaluation

The purpose of our experiments was to get an insight into the behaviour of 4CRULES and assess the feasibility of the optimisation approach for rules processing. Experiments were carried out using an implementation of 4CRULES based on Choco, version 2.1, a Java library for constraint programming systems (<http://choco.sourceforge.net/>).

We devised our experimental model based on practical knowledge of feature catalogues and context models to assess the impact of input context-records on response time and rules applicability. To do so, we created a context model, a catalogue, a rule-set and context records. The context model has 4 context dimensions of maximum domain size 12, which is realistic for consumer applications. The catalogue includes 25 features and 125 precedence constraints and was randomly generated as specified in [18]. The generated catalogue is similar in size to those found in academic literature [17] or used commercially [13]. The rule-set includes 50 FCR which probably exceeds the number of situations a user might really want to “control”. The rules were generated as follows: each rule antecedent was

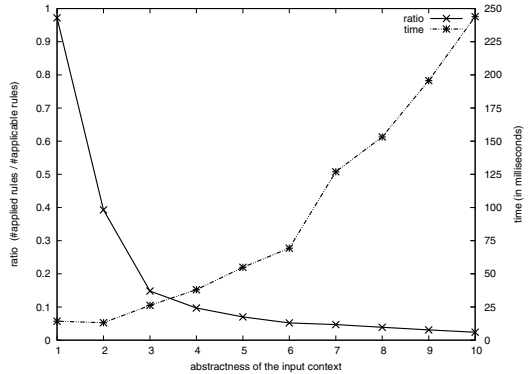


Fig. 4. Behaviour of 4CRULES wrt. input context-records

generated by randomly selecting 2 concrete values per dimension, and each consequent was generated by randomly selecting 2 features. The same priority was used for all rules. For i ranging from 1 to 10, we generated a context-record containing i concrete values per dimension.

We ran the rules engine with each context-record using the rule-set and the catalogue. The results are shown in Figure 4. The x-axis represents the abstractness of the context-record, that is, the parameter i . The y-axis depicts as the ratio of the number of applied rules to the number of applicable rules (left axis) as well as the response time (in milliseconds) for finding an optimal set of rules (right axis). Each point in the plot is an average of 25 instances. The graph shows that as the abstractness of the context-record increases the response time increases while the ratio of applicability decreases. These results confirm the practicality of the proposed model in terms of response time.

10 Conclusion

We have introduced 4CRULES, a rule-based system that enables context-sensitive call control using feature configuration rules. 4CRULES has been designed to be interoperable with standard context services and compositional feature architectures. 4CRULES handles conflicting preferences and mitigates the uncertainty affecting context data. A constraint optimization approach is used to compute configurations that meet the user requirements to an optimum degree.

The approach adopted was to compute optimal consistent rule-sets by determining the mutual applicability of rules to the input context, the compatibility of their configurations with feature interaction constraints, and their aggregate value using a lexicographic ordering combining rule priority, concreteness, and probability of applicability. Experiments on random test instances confirmed the practicality of the approach and highlighted performance critical factors.

Future work will involve the investigation of the rules edition functionalities (e.g., validation, refactoring, compilation), richer ontological models for context domains and catalogues (e.g., feature dependencies), and new application domains (e.g., smart RSS feeds, software plug-in configurations).

Acknowledgments

This material is based upon work supported by the Science Foundation Ireland under Grant numbers 05/IN/I886, 08/PI/I1912, and Embark Post Doctoral Fellowships numbers CT1080049908 and CT1080049909.

References

1. Calder, M., Kolberg, M., Magill, E.H., Reiff-Marganiec, S.: Feature Interaction: A Critical Review and Considered Forecast. *Computer Networks* 41(1), 115–141 (2003)
2. Bond, G.W., Cheung, E., Purdy, H., Zave, P., Ramming, C.: An Open Architecture for Next-Generation Telecommunication Services. *ACM Transactions on Internet Technology* 4(1), 83–123 (2004)

3. Lesaint, D., Papamargaritis, G.: Personalised Communications. In: Voudouris, C., Owusu, G., Dorne, R., Lesaint, D. (eds.) *Service Chain Management - Technology Innovation for the Service Business*, pp. 187–203. Springer, Heidelberg (2008)
4. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.B., Peterson, J., Sparks, R., Handley, M., Schooler, E.M.: SIP: Session Initiation Protocol. In: RFC 3261, IETF (June 2002)
5. Sparks, R.: SIP: Basics and Beyond. *ACM Queue* 5(2), 22–33 (2007)
6. Lennox, J., Wu, X., Schulzrinne, H.: Call Processing Language (CPL): A Language for User Control of Internet Telephony Services. RFC 3880, IETF (October 2004)
7. Wu, X., Schulzrinne, H.: Handling Feature Interactions in the Language for End System Services. In: *Feature Interactions in Telecommunications and Software Systems VIII (ICFI 2005)*, Leicester, UK, pp. 28–30. IOS Press, Amsterdam (June 2005)
8. Turner, K.J., Reiff-Marganiec, S., Blair, L., Pang, J., Gray, T., Perry, P., Ireland, J.: Policy Support for Call Control. *Computer Standards & Interfaces* 28(6), 635–649 (2006)
9. Reiff-Marganiec, S., Turner, K.J., Blair, L.: APPEL: The ACCENT Project Policy Environment/Language. Technical report, University of Stirling, Scotland (December 2005)
10. Blair, L., Turner, K.J.: Handling Policy Conflicts in Call Control. In: Reiff-Marganiec, S., Ryan, M. (eds.) *Feature Interactions in Telecommunications and Software Systems VIII, ICFI 2005*, Leicester, UK, June 2005, pp. 39–57. IOS Press, Amsterdam (2005)
11. Jackson, M., Zave, P.: Distributed Feature Composition: a Virtual Architecture for Telecommunications Services. *IEEE Transactions on Software Engineering* 24(10), 831–847 (1998)
12. Jackson, M., Zave, P.: *The DFC Manual*. AT&T (November 2003)
13. Bond, G.W., Cheung, E., Goguen, H., Hanson, K.J., Henderson, D., Karam, G.M., Purdy, K.H., Smith, T.M., Zave, P.: Experience with Component-Based Development of a Telecommunication Service. In: Heineman, G.T., Crnković, I., Schmidt, H.W., Stafford, J.A., Szyper-ski, C., Wallnau, K. (eds.) *CBSE 2005*. LNCS, vol. 3489, pp. 298–305. Springer, Heidelberg (2005)
14. Zave, P.: An Experiment in Feature Engineering. In: McIver, A., Morgan, C. (eds.) *Programming Methodology*, pp. 353–377. Springer, Heidelberg (2003)
15. Zave, P., Cheung, E.: Compositional Control of IP Media. In: Diot, C., Ammar, M., da Costa, C.S., Lopez, R., Leitao, A.R., Feamster, N., Teixera, R. (eds.) *Proc. of the 2nd Conf. on Future Networking Technologies (CoNext 2006)*, Lisboa, Portugal, SIGCOMM, pp. 67–78 (December 2006)
16. Zave, P.: Audio Feature Interactions in Voice-over-IP. In: Bond, G.W., Schulzrinne, H., Sisalem, D. (eds.) *Proc. of the 1st Int. Conf. on Principles, Systems and Applications of IP Telecommunications (IPTComm)*, New York, NY, pp. 67–78 (July 2007)
17. Zimmer, A.P.: *Prioritizing Features Through Categorization: An Approach to Resolving Feature Interactions*. PhD thesis, University of Waterloo, Canada (September 2007)
18. Lesaint, D., Mehta, D., O’Sullivan, B., Quesada, L., Wilson, N.: Personalisation of Telecommunications Services as Combinatorial Optimisation. In: *IAAI 2008*, pp. 1693–1698. AAAI Press, Menlo Park (2008)

Load Balancing and Almost Symmetries for RAMBO Quorum Hosting*

Laurent Michel¹, Alexander A. Shvartsman¹,
Elaine Sonderegger¹, and Pascal Van Hentenryck²

¹ University of Connecticut, Storrs, CT 06269-2155

² Brown University, Box 1910, Providence, RI 02912

Abstract. RAMBO is the Reconfigurable Atomic Memory for Basic Objects, a formally specified algorithm that implements atomic read/write shared memory in dynamic networks, where the participating hosts may join, leave, or fail. To maintain availability and consistency in such dynamic settings, RAMBO replicates objects and uses quorum systems that can be reconfigured in response to perturbations in the environment. This is accomplished by installing new quorum configurations and removing obsolete configurations, while preserving data consistency. Given the dynamic nature of the atomic memory service, it is vitally important to reconfigure the system online, while making well-reasoned selections of new quorum configurations. This paper reexamines the quorum hosting problem, concentrating on better load balancing models and a novel use of *almost symmetries* for breaking similarities among hosts in the target network. The resultant performance improvements allow more reasonably-sized systems to be reconfigured online in a way that optimizes hosting of quorums with respect to relevant performance criteria.

1 Introduction

Providing consistent shared objects in dynamic networked systems is one of the fundamental problems in distributed computing. Shared object systems must be resilient to failures and guarantee consistency despite the dynamically changing collections of hosts that maintain object replicas. RAMBO, which stands for Reconfigurable Atomic Memory for Basic Objects [6,4], is a formally specified distributed algorithm designed to support a long-lived atomic read/write memory service in such a rapidly changing network environment. To maintain availability and consistency of the memory service, RAMBO uses reconfigurable quorum systems, where each object is replicated at host computers that are quorum members. The intersection among quorum sets is used to guarantee atomicity (linearizability) of the replicated data objects, thereby ensuring behavior equivalent to that of a centralized shared memory.

The ability to rapidly reconfigure quorum systems in response to failures and delays is at the heart of the RAMBO service. Any participant may request a new

* This work was partially supported through NSF awards CCF-0702670 and IIS-0642906 and AFOSR Contract FA955007C0114.

configuration of quorum system hosts, after which consensus is used to agree on the new configuration to be installed. While the RAMBO service permits any new quorum configuration to be installed on-the-fly, it is important to install configurations that benefit system performance. To this end, the quorums should be composed of members who have been communicating with low latency, and be well-balanced with respect to read and write operation loads. Configuration selection must be done quickly, since a lengthy process may impact the liveness and fault-tolerance of the system, as hosts may continue to fail and their observed performance characteristics change over time.

The models in [7] focused on determining optimal quorum configurations. Both constraint programming and local search techniques were used to demonstrate the feasibility of finding high-quality configurations that positively affect the performances of read and write operations and the liveness of the system. CP and hybrid CP/CBLS models were implemented with COMET [14,15,8].

This work began by studying the optimal quorum configurations found in [7] to better understand their properties. Patterns emerged for the attributes of good quorum configurations and for the relationships between quorum systems and their hosting network topologies. The insights led to a significantly better constraint programming model for the RAMBO quorum hosting problem.

While [7] offered a solution to an open problem, the benchmarks were based on modestly sized quorum systems, and it was debatable whether the approach would scale. The contributions of this paper address these questions. First, the paper revisits the way load-balancing is modeled to deliver a more realistic and robust model. Second, the paper offers a decomposition-based model that separately optimizes the replica deployment with respect to the induced delays and the quorum selection with respect to the balancing objective. Third, the paper expands on the symmetry-breaking techniques found in [7] and exploits a dominance relation also known as an “almost symmetry” [5,11]. The dominance is handled with a dynamic symmetry breaking embedded in the search of the first phase of the decomposition. Finally, the paper presents experimental results demonstrating that the extensions are transformative, bringing orders of magnitude in performance improvements (up to 10,000 times faster) and addressing the scalability issues for network topologies endowed with symmetrical structures. Although this work is presented in the context of RAMBO, the results can be applied to any distributed service that relies on dynamically introduced quorum systems.

Section 2 presents RAMBO and quorums in more detail. Section 3 introduces a high-level model for the quorum hosting problem, and Section 4 presents the CP models. Section 5 reports the experimental results, and Section 6 concludes.

2 RAMBO and Quorums

RAMBO, like most data replication services for distributed systems, uses quorums to ensure the atomicity of its data in the presence of failures. Quorum systems [3,13] are collections of sets, called quorums, whose members are hosts

that maintain replicas of the data; in a quorum system any two quorums have a non-empty intersection. If each operation contacts a complete quorum of hosts containing data replicas, then any operation is guaranteed to see the results of the most recently completed operation because there must be at least one host in the intersection of the two respective quorums that participate in the operations. RAMBO uses a variant, called read/write quorum systems, where such a system has two sets of quorums, read quorums and write quorums, with each read quorum having a non-empty intersection with every write quorum.

Figure 1 shows four read/write quorum systems. *Maj* consists of six members grouped into four read quorums and four write quorums, each of which is a majority quorum [13]. For *Wheel*, *3x3*, and *4x4*, the horizontal ovals represent the read quorums, and the more vertical shapes represent the write quorums.

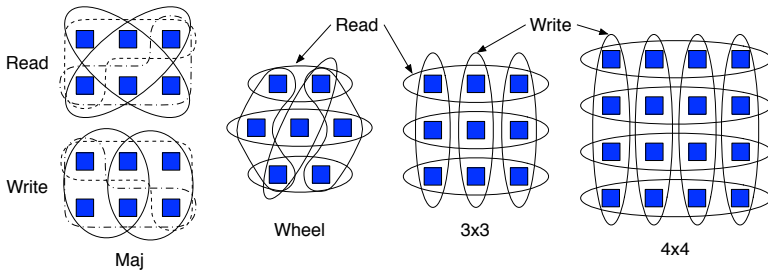


Fig. 1. Examples of Read/Write Quorum Systems

The read and write operations in RAMBO use a two-phase strategy. The first phase gathers information from at least one read quorum of all active configurations, and the second phase propagates information to at least one write quorum of each active configuration.

What sets RAMBO apart from other data replication services is its ability to dynamically reconfigure the quorum system as hosts join, leave, and fail. Quorum reconfiguration is performed concurrently with read and write operations. As long as the members of at least one read quorum and one write quorum of the active configurations are still functioning, reconfiguration can take place. The *4x4* quorum system, for example, can tolerate three failures, but not necessarily four. Thus, speed of reconfiguration is paramount as failures and changes in participants are detected.

RAMBO configurations are “hosted” quorum systems. To optimize configurations, this paper distinguishes between a configuration’s abstract “unhosted” quorum system and its assignment of quorum members to host computers in the target network. The RAMBO *quorum hosting problem* is to assign each member of a given quorum system to a participating host in such a way that the total delays for hosts to contact read and write quorums are minimized. Once an assignment is computed, the resulting configuration is augmented to include information recommending the best read and write quorums for each host to

use, where use of the best quorums will result in minimum delays for read and write operations and the most balanced load on replica hosts (until failures occur in those best quorums). Of course, the use of this information is optional, and a host can always propose another quorum system if it does not observe good responses from the recommended system.

The selection of a new configuration should be made dynamically in response to external stimuli and observations about the performance of the service. RAMBO participants have no knowledge of the underlying network, particularly as nodes join and leave. Hence, each host measures (externally to RAMBO) its average message delays with every other host as the best available estimate of network connections and host failures. Each host also measures the average frequency of its read and write operations. The gathered information is shared with the other hosts by piggy-backing this information onto routine messages of RAMBO. The overall guiding principle is that observations of current behaviors are the best available predictors of future behaviors.

3 Modeling the RAMBO Quorum Hosting Problem

Model Parameters. The inputs for the RAMBO quorum hosting model are:

- The set of hosts H .
- For every host $h \in H$, the average frequency f_h of its read and write requests.
- For every pair of hosts $h_1, h_2 \in H$, the average round trip delay d_{h_1, h_2} of messages from h_1 to h_2 .
- The quorum system to be deployed on H , which consists of:
 - The set of members M , whose hosts will maintain replicas of the data.
 - The set of read quorums $R \subseteq \mathcal{P}(M)$.
 - The set of write quorums $W \subseteq \mathcal{P}(M)$.

The quorum system must be well-formed, meaning $\forall r \in R, \forall w \in W, r \cap w \neq \emptyset$.

Decision Variables. A decision variable x_m with domain H is associated with each quorum system member m . $x_m = h$ when member m is deployed on host h . Each host h also is associated with two decision variables $readQ_h$ and $writeQ_h$ (with domains R and W) denoting, respectively, one read (write) quorum from the minimum average delay read (write) quorums associated with h . Finally, three auxiliary variables $readLoad_m$, $writeLoad_m$, and $load_m$ represent the read, write, and total loads on replica m that are induced by the traffic between the hosts and their chosen read/write quorums.

The Objective. An optimal quorum hosting minimizes

$$\sum_{h \in H} f_h \times \left(\min_{q \in R} \left(\max_{m \in q} d_{h, x_m} \right) + \min_{q \in W} \left(\max_{m \in q} d_{h, x_m} \right) \right)$$

where each term in the summation captures the time it takes in RAMBO for a host h to execute the read/write phase of the protocol. Indeed, a read (or

a write) in RAMBO requires the client to contact all the members of at least one read quorum before it can proceed to a write round of communication with all the members of at least one write quorum to update the data item. The $\max_{m \in q} d_{h,x_m}$ reflects the time it takes to contact all the members of quorum q as one must wait for an answer from its slowest member. The outer $\min_{q \in R}$ reflects the fact that RAMBO must hear back from one quorum before it proceeds, and this happens when the “fastest” quorum replies.

The Constraints. A quorum hosting is subject to the following constraints. First, all quorum system members must be deployed on separate hosts.

$$\forall m, m' \in M : m \neq m' \Rightarrow x_m \neq x_{m'}$$

An implementation of RAMBO may use different strategies when contacting the read and write quorums. A conforming implementation might simply contact all the read quorums in parallel. Naturally, this does not affect the value of the objective, but it induces more traffic and work on the hosts of quorum members. Another strategy for RAMBO is to contact what it currently perceives as the fastest read quorum first and fall back on the other read quorums if it does not receive a timely response. It could even select a read quorum uniformly at random. These strategies strike different tradeoffs between the traffic they induce and the workload uniformity. The model presented below captures the greedy strategy, namely, RAMBO contacts its closest quorum first, and the model assumes that this quorum replies (unless a failure has occurred).

The model uses the $readQ_h$ and $writeQ_h$ variables of host h to capture which read (write) quorum RAMBO contacts. Recall that a read (write) quorum is a set and therefore the domain of $readQ_h$ is the set of read quorums (similarly for $writeQ_h$). More formally,

$$readQ_h = r \Rightarrow \max_{m \in r} d_{h,x_m} = \min_{q \in R} \left(\max_{m \in q} d_{h,x_m} \right)$$

$$writeQ_h = w \Rightarrow \max_{m \in w} d_{h,x_m} = \min_{q \in W} \left(\max_{m \in q} d_{h,x_m} \right)$$

Each equation requires the chosen quorum to induce a minimal delay. Note that several quorums might deliver the same minimal delay, so this constraint alone does not determine a host’s ideal read (write) quorum.

The third set of constraints defines the read, write, and total loads of a quorum system member as

$$readLoad_m = \sum_{h \in H} \sum_{m \in readQ_h} f_h$$

$$writeLoad_m = \sum_{h \in H} \sum_{m \in writeQ_h} f_h$$

$$load_m = readLoad_m + writeLoad_m$$

Clearly, the loads on m depend on which read and write quorums are chosen among those that induce minimal delays.

3.1 Load Balancing

For load balancing, the RAMBO deployment model in [7] uses an additional input parameter α limiting the spread of loads by requiring the maximum load on a quorum system member to be within a factor α of the minimum load.

One insight, driving the development of the new load-balancing models, is that in order for a quorum system member to be useful in a data replication system, not only must its total load be non-zero, but both its read and write loads also must be non-zero. If a quorum system member is never used in a write quorum, it can never have the most recent data value, and if a quorum system member is never used in a read quorum, it does not matter whether or not it has the most recent data value. This pathological setup is easily avoided by requiring $readLoad_m > 0$ and $writeLoad_m > 0$.

Carrying this reasoning further, it is inappropriate to allow the load on a quorum system member to be predominantly read requests or predominantly write requests as is possible when only the total load on members is balanced. Instead, read and write loads should be balanced separately.

The first model to consider uses the load factor α to separately constrain the read and write loads. Unfortunately, this is not a sensible approach since, for small values of α , many networks do not have satisfying quorum hostings. This is particularly true when a few hosts send the bulk of the messages.

New Load-Balancing Model. The adopted approach uses two optimizations, rather than a single optimization, to obtain a balanced hosting. The first optimization finds an assignment of quorum system members to hosts that minimizes communication delays. The second, given a global optimum of the first, finds an assignment of quorums to hosts that minimizes the load imbalances among quorum system members. Compared to [7], this approach trades off load-balancing among quorum members for faster quorum response times. The loads may be even more balanced for some networks with this approach, however, because the optimization does not stop with an assignment that satisfies the α load-balancing factor. Note that the first optimization only delivers *one* global optimum (when there might be several), and this specific solution might not lead to the most balanced solution in the second optimization.

Two alternative load-balancing objectives are studied. The first minimizes the differences in the read and write loads between the most heavily loaded and the most lightly loaded quorum system members (or alternatively, minimizes α).

$$\min \left(\max_{m \in M} readLoad_m - \min_{m \in M} readLoad_m \right)$$

$$\min \left(\max_{m \in M} writeLoad_m - \min_{m \in M} writeLoad_m \right)$$

The second minimizes the standard deviations for the read and write loads.

$$\min \left(|M| \cdot \sum_{m \in M} (readLoad_m)^2 - \left(\sum_{m \in M} readLoad_m \right)^2 \right)$$

$$\min \left(|M| \cdot \sum_{m \in M} (\text{writeLoad}_m)^2 - \left(\sum_{m \in M} \text{writeLoad}_m \right)^2 \right)$$

The second objective yields more middle-of-the-range loads for quorum system members, but possibly a slightly larger range of values. Minimizing the standard deviation of loads was found to be an effective technique for balancing work loads among nurses [11], but its added cost may not be justifiable for this application.

3.2 Network Symmetries

Many network topologies have some symmetries among host nodes. Ideally, these symmetries can be exploited in determining optimal quorum placements. Consider, for example, the partial network illustrated in Figure 2. Hosts *B*, *C*, *D*, *E*, and *F* have a single neighboring host *A*. To the rest of the network beyond *A*, shown with three groups of \dots , hosts *B* through *F* are equivalent because they all are the same number of “hops” away. The maximum delay for these unillustrated hosts to access a quorum consisting of hosts *A* and *B*, represented by the solid oval, is the delay to get a response from *B*. Similarly, the maximum delay to access a quorum of *A* and *C*, represented by the dashed oval, is the delay to get a response from host *C*. Since both *B* and *C* are one hop beyond *A*, these delays are approximately the same, and the two quorums are equivalent for the hosts beyond *A*. The two quorums also are equivalent for host *A*.

The two quorums are not equivalent for hosts *B* and *C*, however. If *B* uses the quorum with hosts *A* and *B*, its maximum delay to access the quorum is the time it takes to get a response from *A*, whereas if *B* uses the quorum with hosts *A* and *C*, it also must wait to get a response from *C* which is another hop away. Thus, the quorum with hosts *A* and *B*, represented with the solid oval, is a better quorum for *B* to use. For *C*, the better quorum is the one with hosts *A* and *C*, represented with a dashed oval.

An optimal quorum hosting minimizes the system’s total communication for accessing quorums, where each host’s contribution to the total delay is its message frequency times the delays to contact its best read and write quorums. Assume *B* has a frequency of 10, and *C* has a frequency of 5. Then the overall objective would be less using the solid quorum with *A* and *B*, rather than the dashed quorum with *A* and *C*, because *B*’s message frequency is greater than *C*’s.

The relationship among hosts *B* through *F* is an almost symmetry [51], rather than a true symmetry. The hosts are equivalent with respect to hosts outside the group. Within the group, the frequencies impose a dominance relation among hosts which requires special care in the search. Dominance in CP was studied in [10], while almost symmetries received some attention for planning [9] and graphs [2]. This realistic application demonstrates their true potential.

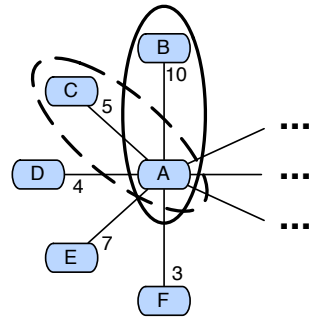


Fig. 2. Network Topology with an Almost Symmetry

4 The CP Model for the Quorum Hosting Problem

The initial COMET program for the quorum hosting problem is shown in Figure 3. The data declarations in lines 2–8 correspond to the input data of the RAMBO model in Section 3. Line 9 declares an additional input used for breaking variable symmetries among the members of the quorum system. Lines 10–14 define derived data. Specifically, $nbrQ[m]$ is the number of quorums in which m appears, and $degree[h]$ is the number of immediate neighbors of host h in the network, as determined from the observed message delays. RQ and WQ are the index sets of the read and write quorums, respectively. The auxiliary matrices $readQC$ and $writeQC$ are encodings of quorum membership, e.g., $readQC[i, j] = true \Leftrightarrow j \in R[i]$.

Lines 15–22 declare the decision variables. Variable $x[m]$ specifies the host of quorum system member m . Variables $readD[h, r]$ and $writeD[h, w]$ are the communication delays for host h to access read quorum r and write quorum w . The variables $readQ[h]$ and $writeQ[h]$ represent the read and write quorum selections for host h . Finally, the variables $readLoad[m]$ and $writeLoad[m]$ represent the read and write communication loads on quorum system member m , given the current deployment and quorum selections, and $load[m]$ represents the total communication load on member m . Note that the domains for $readLoad[m]$ and $writeLoad[m]$ exclude zero.

Line 24 specifies the objective function, which minimizes the total communication delay over all operations. Line 26 specifies the fault tolerance requirement, namely, all members of the quorum system must be deployed to distinct hosts. The `onDomains` annotation indicates that arc-consistency must be enforced. Line 27 breaks the variable symmetries among the quorum system members [12].

Lines 28–39 constrain the auxiliary delay variables and quorum selection variables needed in the load-balancing constraint. The constraints on lines 28 and 29 capture the delays incurred by host h to use a read (write) quorum. Lines 31 and 32 require the quorums assigned to host h , namely $readQ[h]$ and $writeQ[h]$, to be among the quorums with minimum delay for that host. Lines 35–37 specify the read, write, and total communication loads on m as the sum of the operation frequencies of each host for which m is a member of its assigned read and/or write quorum. Line 39 is the load-balancing constraint requiring the load on the most heavily loaded quorum system member to be no more than α times the load on the most lightly loaded member.

The search procedure operates in two phases. The first phase (lines 41–45) assigns quorum system members to hosts. The variable selection heuristic first focuses on members that appear in many quorums, and the value selection heuristic first considers hosts that have many neighbors “close by” as these would be ideal locations for data replicas. The second phase (lines 46–49) finds an assignment of hosts to read and write quorums that satisfies the load-balancing constraint. This second phase cannot impact the value of the objective function. Rather, its role is to decide which quorum each host should use to meet the load-balancing requirement. Clearly, only one such assignment is needed which explains the `once<cp>` annotation on line 46. Lines 46–49 consider the most

```

1 Solver<CP> cp();
2 range M= ...; // The members of the quorum system
3 set{int}[] R = ...; // An array storing all the read quorums of the quorum system
4 set{int}[] W = ...; // An array storing all the write quorums of the quorum system
5 range H = ...; // The host nodes
6 int[] f = ...; // The frequency matrix
7 int[,] d = ...; // The delays matrix
8 int alpha = ...; // The load factor
9 set{tuple{int low; int high}} Order = ...; // The order of quorum members
10 int nbrQ[m in M] = ...; // The number of quorums for each member
11 int degree[H] = ...; // The degree of a host (number of neighbors)
12 range RQ = R.getRange(); range WQ = W.getRange();
13 boolean readQC[RQ,M] = ...;
14 boolean writeQC[WQ,M] = ...;
15 var<CP>{int} x[M](cp,H);
16 var<CP>{int} readD[H,RQ](cp,0..10000);
17 var<CP>{int} writeD[H,WQ](cp,0..10000);
18 var<CP>{int} readQ[H](cp,RQ);
19 var<CP>{int} writeQ[H](cp,WQ);
20 var<CP>{int} readLoad[M](cp,1..10000);
21 var<CP>{int} writeLoad[M](cp,1..10000);
22 var<CP>{int} load[M](cp,0..10000);
23 minimize <cp>
24 sum(h in H) f[h] * (min(r in RQ) readD[h,r] + min(w in WQ) writeD[h,w])
25 subject to {
26   cp.post(alldifferent(x), onDomains);
27   forall(o in Order) cp.post(x[o.low] < x[o.high]);
28   forall(h in H,r in RQ) cp.post(readD[h,r] == max(m in R[r]) d[h,x[m]]);
29   forall(h in H,w in WQ) cp.post(writeD[h,w] == max(m in W[w]) d[h,x[m]]);
30   forall(h in H) {
31     cp.post(readD[h,readQ[h]] == min(r in RQ) readD[h,r]);
32     cp.post(writeD[h,writeQ[h]] == min(w in WQ) writeD[h,w]);
33   }
34   forall(m in M) {
35     cp.post(readLoad[m] == sum(h in H) f[h] * readQC[readQ[h],m]);
36     cp.post(writeLoad[m] == sum(h in H) f[h] * writeQC[writeQ[h],m]);
37     cp.post(load[m] == readLoad[m] + writeLoad[m]);
38   }
39   cp.post(max(m in M) load[m] <= alpha * min(m in M) load[m]);
40 } using {
41   while (sum(k in M) x[k].bound() < M.getSize())
42     selectMax(m in M: !x[m].bound()) (nbrQ[m])
43     tryall<cp>(h in H : x[m].memberOf(h)) by (- degree[h])
44       cp.label(x[m], h);
45     onFailure cp.diff(x[m], h);
46 once<cp> forall(h in H : !readQ[h].bound() || !writeQ[h].bound()) by (- f[h]) {
47   label(readQ[h]);
48   label(writeQ[h]);
49 }
50 }

```

Fig. 3. The Initial CP Model in COMET

“talkative” hosts first (by decreasing frequencies) and attempt to assign one of the remaining legal (minimal delay) quorums from its domain.

Improvements to the Search Heuristic. Within the search, the variable selection heuristic focuses on members that appear in many quorums. This works well for quorum systems such as **Maj** and **Wheel** in Figure 1 where some members are in more quorums than others, but it doesn’t work well for systems such as **3x3** and **4x4** where every member is in exactly two quorums. A better search heuristic for **3x3** and **4x4** exploits the symmetries in the quorum system by focusing on variables with smaller domains. The following search heuristic, which replaces line 42 in Figure 3, combines both goals.

```
1 selectMin(m in M: !x[m].bound()) (x[m].getSize() - 4 * nbrQ[m])
```

The factor of 4 is somewhat arbitrary, but its intent is to give more weight to small differences in the number of quorums to which a member belongs.

Load-Balancing Improvements. Because the read and write loads are balanced separately, the variables *readLoad[m]* and *writeLoad[m]* are independent of each other. Thus, the search can be improved by replacing lines 46–49 in Figure 3 with

```
1 once<cp> forall(h in H : !readQ[h].bound()) by (- f[h])
2   label(readQ[h]);
3 once<cp> forall(h in H : !writeQ[h].bound()) by (- f[h])
4   label(writeQ[h]);
```

By dividing the single **once**<cp> block into two, the need to backtrack over a satisfactory assignment of read quorums to hosts while searching for a satisfactory assignment of write quorums to hosts is eliminated.

To optimize the load balances, the load-balancing constraint on line 39 is deleted, and the optimization code is appended after line 50. Figure 4 contains the code for optimizing the load balance for read operations; a similar fragment optimizes the load balance for write operations with *writeLoad[m]*.

Figure 4 begins with the definition of a new solver *cpR* for read-load balancing. Line 2 defines *bestR* as the difference between the heaviest and lightest read loads using the quorum hosting assignments from the communication optimization. New decision variables are declared in lines 3–5. Variable *objR* is the objective to be minimized and starts off with an upper bound equal to *bestR*. Variables *readQ2[h]* and *readLoad2[m]* correspond to the variables *readQ[h]* and *readLoad[m]* from Figure 3.

The objective function, specified in lines 6–8, minimizes *objR*, the difference between the heaviest and lightest read loads using the load-balancing quorum-to-host assignments. The constraints on lines 10 and 12 define *readQ2[h]* to be one of the read quorums with the fastest response for host *h* and *readLoad2[m]* to be the resultant load on quorum system member *m*. These constraints mirror the constraints on lines 31 and 35 of Figure 3. Finally, lines 14 and 15 assign read quorums to hosts, beginning with the most “talkative” hosts.

```

1 Solver<CP> cpR();
2 int bestR = max(m in M) readLoad[m] - min(m in M) readLoad[m];
3 var<CP>{int} objR(cpR,0..bestR);
4 var<CP>{int} readQ2[H](cpR,RQ);
5 var<CP>{int} readLoad2[M](cpR,1..10000);
6 minimize <cpR> objR
7 subject to {
8   cpR.post(objR == max(m in M) readLoad2[m] - min(m in M) readLoad2[m]);
9   forall(h in H)
10    cpR.post(readD[h,readQ2[h]] == min(q in RQ) readD[h,q]);
11   forall(m in M)
12    cpR.post(readLoad2[m] == sum(h in H)(f[h] * readQC[readQ2[h],m]));
13 } using {
14   forall(h in H : !readQ2[h].bound()) by (- f[h])
15     label(readQ2[h]);
16 }

```

Fig. 4. Solver for Optimally Balancing Read Loads

To minimize the standard deviation of loads, line 2 of Figure 4 is replaced by

```

1 int bestR = M.getSize() * sum(m in M)(readLoad[m] ^ 2)
2   - (sum(m in M) readLoad[m]) ^ 2;

```

and line 8 is replaced with

```

1   cpR.post(objR == M.getSize() * sum(m in M)(readLoad2[m] ^ 2)
2     - (sum(m in M) readLoad2[m]) ^ 2);

```

Note that the `spread` global constraint cannot be used since the total load over all quorum system members depends on the selected quorums for each host.

Breaking Almost Symmetries. Figure 5 shows the changes to break the almost symmetries in the topology, where lines 6–15 replace lines 42–45 in Figure 3.

Lines 1–3 contain additional parameter declarations. Eq is the set of sets of equivalent hosts used in the value symmetry breaking. The sets of immediate neighbors for each host are computed first using the observed message delays, and then two hosts are deemed equivalent placements for a quorum system member if they have the same set of neighbors. The set $notEq$ contains all the hosts not in some set of Eq . For each quorum system member m , the variable $minQSz[m]$ is the size of smallest quorum of which m is a member.

The set $searchH$ guides the search. $domain$, on line 7, is the set of hosts currently in the domain of the selected member m , and $searchH$, on lines 8–11, is the subset of hosts currently being considered for assignment to m . $searchH$ contains all the hosts in the domain of m that are not equivalent to any other host, plus one additional unbound host from each equivalence set, if one exists.

Because the host symmetries are *almost symmetries* rather than true symmetries, care must be taken both in the order in which hosts are added to

```

1 set{set{int}} Eq = ...; // The sets of equivalent hosts
2 set{int} notEq = ...; // The non-equivalent hosts
3 int minQsz[m in M] = ...; // The smallest quorum size for each member
4
5 while (sum(k in M) x[k].bound() < M.getSize())
6   selectMin(m in M: !x[m].bound())(minQsz[m], x[m].getSize()-4*nrQ[m]) {
7     set{int} domain = collect(h in H : x[m].memberOf(h)) h;
8     set{int} searchH = notEq inter domain;
9     forall(e in Eq : card(e inter domain) > 0)
10      selectMax(en in e inter domain) (f[en])
11      searchH.insert(en);
12    tryall<cp>(h in searchH) by (- degree[h])
13      cp.label(x[m], h);
14    onFailure cp.diff(x[m], h);
15  }

```

Fig. 5. Search Procedure with Almost Symmetry Breaking

searchH and in the order in which quorum system members are selected for binding. Clearly, hosts should be added to *searchH* beginning with the hosts with the highest message frequencies, as on line 10. The more subtle requirement is that quorum system members must be assigned in increasing quorum size order, as specified on line 6. To minimize the communication delays for the most “talkative” hosts, those hosts should be only one “hop” away from the members of their best read and write quorums, if possible. A host is zero hops from itself, but two hops from the other hosts in its equivalence set. For small quorums, this difference in hops is more likely to impact the delay to contact a complete quorum. Hence, the members of small quorums must be assigned first.

5 Experimental Results

The Benchmarks. The benchmarks represent characteristics of common network topologies and quorum systems. As illustrated in Figure 6, **Stars3**, **Stars2**, and **Stars2c3** arrange 15 hosts in clusters, **Line** arranges 15 hosts in a single line, and **Switch** consists of 10 hosts on a switch and 4 other hosts hooked up via point-to-point links. **Hyper16** interconnects 16 hosts in a hypercube. Three larger benchmarks, **BigStars3**, **BigStars2**, and **BigStars2c3**, are 21-host extensions of **Stars3**, **Stars2**, and **Stars2c3**, with the extra hosts added to the “stars”. Figure 6 shows the frequencies of the read/write operations for each host; the delays are the number of “hops” between hosts. The quorum system benchmarks are illustrated in Figure 11.

Model Comparisons. Table 11 compares results for different quorum hosting models using COMET 2.1 on a Core 2 at 2.4 GHz. Columns are grouped by model type; the first column uses the model from 7, the second column adds non-zero load constraints and separate load optimization, the third column is the

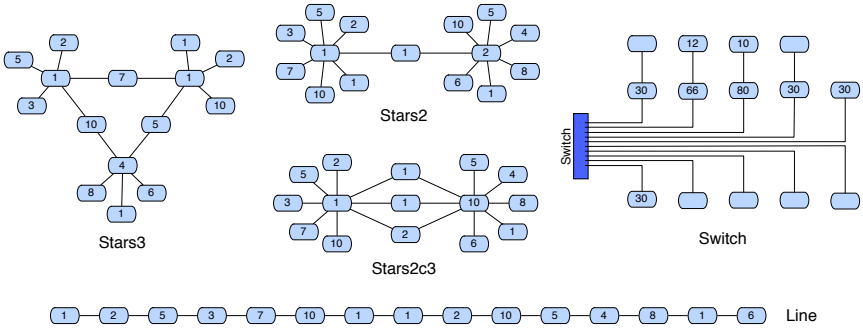


Fig. 6. Network Benchmarks Stars3, Stars2, Stars2c3, Switch, and Line

full load-balancing model, and the fourth column adds almost-symmetry breaking. Within each group, column *Opt* gives the objective for the optimal solution found, T_{end} gives the time in seconds to prove optimality, and column T_{opt} reports the time in seconds to find the optimum. The table provides two rows for each benchmark: the first reports the average and the second reports the standard deviation over 50 runs. The 3x3 quorum system is used throughout.

It is useful to review these results in more detail.

1. For Stars2c3 the symmetry model is over 10,000 times faster than [7].
2. Exploiting almost symmetries in network topologies significantly impacts performance, particularly for Stars2 and Stars2c3 that have many symmetric hosts. Note neither Line nor Hyper16 have network symmetries.
3. Hyper16 has more balanced loads with the new models; for the other benchmarks the loads often are less balanced but the objective is smaller, reflecting an improved tradeoff between response times and load balancing.

Table 1. Experimental Results Comparing Models with 3x3 Quorum System

Benchmark	Original Model From [7]			Separate Load Optimization			Independent Read & Write			Symmetry Breaking			
	<i>Opt</i>	T_{end}	T_{opt}	<i>Opt</i>	T_{end}	T_{opt}	<i>Opt</i>	T_{end}	T_{opt}	<i>Opt</i>	T_{end}	T_{opt}	
Stars3	μ	284	39.83	12.52	284	43.77	10.30	284	24.51	0.45	284	2.01	0.12
	σ		6.79	12.32		8.47	10.90		0.95	0.58		0.04	0.08
Stars2	μ	316	527.50	163.88	287	116.12	10.41	287	108.50	6.59	287	0.09	0.03
	σ		375.07	366.84		15.37	12.09		5.50	3.63		0.01	0.01
Stars2c3	μ	268	1271.47	800.50	240	29.10	5.48	240	14.98	1.79	240	0.09	0.03
	σ		460.78	648.67		4.91	4.90		0.98	0.86		0.01	0.01
Switch	μ	620	47.57	32.53	620	34.87	19.54	620	3.60	0.01	620	0.35	0.02
	σ		54.31	54.12		28.98	28.60		0.33	0.01		0.03	0.01
Line	μ	517	388.41	269.31	499	156.04	71.66	499	211.05	111.78	499	206.05	109.11
	σ		105.92	156.36		24.12	38.74		3.62	3.31		4.21	2.56
Hyper16	μ	249	438.84	215.74	249	421.40	212.98	249	295.60	57.66	249	294.26	58.15
	σ		54.53	114.21		53.50	115.30		5.93	21.55		4.54	21.65

Table 2. Experimental Results Comparing Load-Balancing Models

		Wheel						Maj					
		Min. Difference			Min. St. Dev.			Min. Difference			Min. St. Dev.		
Benchmark		T_{end}	T_{opt}	$\#c$	T_{end}	T_{opt}	$\#c$	T_{end}	T_{opt}	$\#c$	T_{end}	T_{opt}	$\#c$
Stars3	μ	2.64	0.43	5.0	2.65	0.43	6.0	1.50	0.03	32.4	1.50	0.03	186.6
	σ	0.05	0.29	0.0	0.05	0.29	0.0	0.02	0.01	11.1	0.01	0.01	25.1
Stars2	μ	0.20	0.07	36.0	0.20	0.07	59.0	0.31	0.03	717.2	1.18	0.03	10571.5
	σ	0.01	0.01	0.0	0.01	0.01	0.0	0.04	0.01	271.0	0.29	0.01	2796.8
Stars2c3	μ	0.23	0.02	23.7	0.23	0.02	36.2	0.16	0.02	16.6	0.16	0.02	48.5
	σ	0.01	0.01	46.4	0.01	0.01	112.9	0.01	0.01	8.8	0.01	0.01	10.4
Switch	μ	0.89	0.25	10.0	0.89	0.24	12.0	0.14	0.02	574.2	1.04	0.02	11549.6
	σ	0.02	0.04	0.0	0.01	0.04	0.0	0.04	0.01	562.7	1.16	0.01	13920.2
Line	μ	60.78	29.13	20.2	62.77	30.05	52.3	22.16	4.03	4.6	22.16	4.02	115.5
	σ	2.84	2.40	15.8	2.86	2.57	42.4	1.15	1.12	3.5	1.05	1.18	114.7
Hyper16	μ	80.07	4.93	70.3	81.52	4.97	508.4	53.12	2.74	90.3	54.80	2.81	2234.6
	σ	2.46	0.65	23.4	2.88	0.63	28.9	1.32	0.35	32.3	2.40	0.35	205.4

4. With the new models, an optimal solution is found more quickly, allowing reconfiguration to a good hosting before the proof of optimality completes.

The exception is **Line** which performs poorly with the new search heuristic.

5. The standard deviations for T_{end} and T_{opt} are smaller with the new models.

Load Balancing Options. Table 2 compares two load-balancing options: minimizing the difference between the largest and smallest loads and minimizing the standard deviation among loads. Results are presented for the symmetry breaking model with **Wheel** and **Maj** quorum systems. Within each column group, $\#c$ is the number of choices during load balancing, not the total number of choices.

When adopting the standard-deviation model, the additional delay is small for **Wheel**, but more significant for some of the **Maj** benchmarks. The largest impact is for **BigStars2** with 4×4 , shown in Table 3, which takes an additional 23 seconds to improve the loads from $\{15, 15, 17, 45\}$ and $\{5, 5, 40, 42\}$ to $\{15, 16, 16, 45\}$ and $\{5, 22, 23, 42\}$.

Larger Instances. Table 3 shows the almost-symmetry breaking model extends nicely to larger networks and quorum systems.

Table 3. Experimental Results for Larger Networks with 4×4 Quorum System

Load Balancing		BigStars3				BigStars2				BigStars2c3			
		Opt	T_{end}	T_{opt}	$\#c$	Opt	T_{end}	T_{opt}	$\#c$	Opt	T_{end}	T_{opt}	$\#c$
Min. Difference	μ	436	167.81	45.62	62.1	442	4.66	0.36	376.0	350	1.40	0.14	12.0
	σ		17.14	24.70	7.7		1.14	0.31	0.0		0.13	0.09	0.0
Min. St. Dev.	μ	436	161.68	38.87	696.6	442	27.62	0.34	104020.1	350	1.40	0.15	20.0
	σ		13.61	24.48	122.9		1.70	0.36	15.2		0.13	0.10	0.0

6 Conclusions

Quorums are an effective tool for implementing consistency and availability in replicated distributed data services like RAMBO. The study of optimal quorum configurations led to significant performance improvements to the online quorum hosting model, advancing the solution from the realm of feasible to practical. In particular, better load balancing models and exploitation of almost symmetries in network topologies were presented. While this paper focused on finite-domain models, the improvements are likely to impact CBLs/hybrid models also.

References

1. Donaldson, A.F., Gregory, P.: Almost-Symmetry in Search (SymNet Workshop Proceedings). Technical Report TR-2005-201, University of Glasgow (2005)
2. Fox, M., Long, D., Porteous, J.: Discovering near symmetry in graphs. In: Proceedings of AAAI (2007)
3. Gifford, D.K.: Weighted voting for replicated data. In: Proceedings of the Seventh Symposium on Operating System Principles (SOSP), pp. 150–162 (1979)
4. Gilbert, S., Lynch, N.A., Shvartsman, A.A.: RAMBO II: Rapidly reconfigurable atomic memory for dynamic networks. In: IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 259–268 (2003)
5. Long, D., Fox, M.: Symmetries in planning problems. In: SymCon 2003, CP Workshop (2003)
6. Lynch, N., Shvartsman, A.: RAMBO: A reconfigurable atomic memory service for dynamic networks. In: Proceedings of the 16th International Symposium on Distributed Computing, pp. 173–190 (2002)
7. Michel, L., Moraal, M., Shvartsman, A., Sonderegger, E., Van Hentenryck, P.: Online Selection of Quorum Systems for RAMBO Reconfiguration. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 88–103. Springer, Heidelberg (2009)
8. Michel, L., See, A., Van Hentenryck, P.: Parallelizing constraint programs transparently. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 514–528. Springer, Heidelberg (2007)
9. Porteous, J., Long, D., Fox, M.: The identification and exploitation of almost symmetry in planning problems. In: K. Brown, editor, Proceedings of the 23rd UK Planning and Scheduling SIG (2004)
10. Prestwich, S., Beck, J.C.: Exploiting dominance in three symmetric problems. In: Fourth International Workshop on Symmetry and Constraint Satisfaction Problems, pp. 63–70 (2004)
11. Schaus, P., Van Hentenryck, P., Régim, J.-C.: Scalable Load Balancing in Nurse to Patient Assignment Problems. In: van Hoes, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 193–207. Springer, Heidelberg (2009)
12. Smith, B.M.: Sets of symmetry breaking constraints. In: SymCon, vol. 5 (2005)
13. Thomas, R.H.: A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems* 4(2), 180–209 (1979)
14. Van Hentenryck, P., Michel, L.: *Constraint-Based Local Search*. The MIT Press, Cambridge (2005)
15. Van Hentenryck, P., Michel, L.: Nondeterministic control for hybrid search. *Constraints* 11(4), 353–373 (2006)

Testing Continuous Double Auctions with a Constraint-Based Oracle

Roberto Castañeda Lozano^{1,2}, Christian Schulte¹, and Lars Wahlberg²

¹ KTH – Royal Institute of Technology, Sweden
{rcas,cschulte}@kth.se

² Cinnober Financial Technology AB, Stockholm, Sweden
{roberto.castaneda,lars.wahlberg}@cinnober.com

Abstract. Computer trading systems are essential for today’s financial markets where the trading systems’ correctness is of paramount economical significance. Automated random testing is a useful technique to find bugs in these systems, but it requires an independent system to decide the correctness of the system under test (known as *oracle problem*). This paper introduces a constraint-based oracle for random testing of a real-world trading system. The oracle provides the expected results by generating and solving constraint models of the trading system’s continuous double auction. Constraint programming is essential for the correctness of the test oracle as the logic for calculating trades can be mapped directly to constraint models. The paper shows that the generated constraint models can be solved efficiently. Most importantly, the approach is shown to be successful by finding errors in a deployed financial trading system and in its specification.

1 Introduction

A financial market is a system that allows buyers and sellers to trade and exchange items of value. Nowadays, all major financial markets use computer systems to support their trading activities (1). *TRADEexpress*, developed by Cinnober Financial Technology AB, is an example of such a system. *TRADEexpress* is deployed in different markets around the world, such as the London Metal Exchange, Alpha Trading Systems, or the Hong Kong Mercantile Exchange.

The predominant trading mechanism in financial markets are *continuous double auctions* (2). In a continuous double auction, trade orders are entered continuously and matched against each other as soon as their constraints are satisfied. The numerous trading strategies render the order matching of *TRADEexpress* complex. At the same time, system failures can cause serious economic damage: according to Cinnober’s risk assessment, a failure halting *TRADEexpress* is estimated to incur losses in the order of several million US dollars.

Automated random testing, as a complement to more systematic testing techniques, is useful in finding bugs in trading systems (3). However, it presents an inherent problem, commonly known as the *test oracle problem*: an independent system called *test oracle* is needed to automatically decide the correctness of the

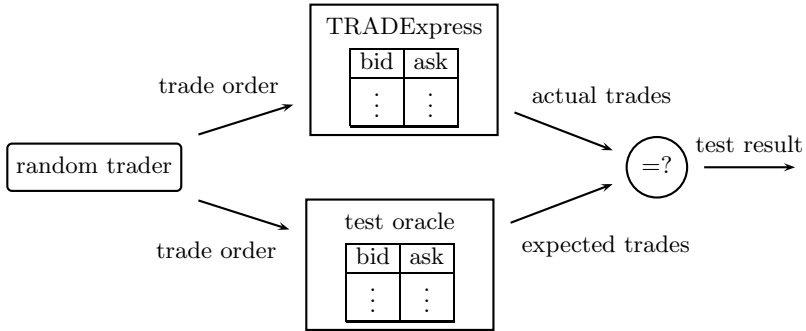


Fig. 1. Random test with the constraint-based test oracle

output generated by the system under test. The usual goals for designing test oracles are correctness, completeness, accuracy, and low development cost (4).

This paper uses constraint programming to solve the test oracle problem that arises for random testing of continuous double auctions. We present a constraint model of the *TRADEExpress*' continuous double auction derived from a set of informal system requirements, and use the model as a test oracle for automated random tests as shown in Fig. 1. The relevance of our approach is illustrated by the number of defects found in the requirements during the modeling process and failures detected during the execution of the random tests.

Related work. Constraint programming has typically been used for automatic test data generation from formal specifications of the system under test (5; 6). Nevertheless, the case where formal specifications are not available has not been thoroughly explored by the constraint programming community, apart from some efforts in the field of hardware testing (7). In the area of financial services, the test oracle problem has typically been approached by developing simple failure detection mechanisms (3). The lack of more complete test oracles has prevented the large-scale application of random testing to trading systems. To the best of our knowledge, there are no previous attempts to design a test oracle that covers the complete functionality of a real-world trading system.

The study of continuous double auctions for other purposes than testing and verification has attracted interest from different areas. Several articles present theoretical models formulated using constraint programming (8) or closely related techniques such as integer programming (9). However, aspects like order priority, which are common issues in most existent trading systems (10, Ch. 2, Appx.), have not been previously considered in these research models.

Main contributions. The main contributions of this paper are as follows:

- a constraint-based model, for the first time, of a non-theoretical continuous double auction;

- results that show the success of a constraint programming approach to the test oracle problem in financial systems, including the detection of several documentation defects and two failures caused by relevant bugs in the system under test;
- empirical evidence of the feasibility and cost-effectiveness of using formal models in the verification of these systems.

Plan of the paper. Section 2 explains how the continuous double auction works in *TRADEexpress*. Section 3 defines the constraint programming model used as a test oracle. Section 4 gives measures related to the main characteristics of the test oracle and results obtained from the execution of random tests. Section 5 concludes and discusses future perspectives.

2 TRADEexpress and the Continuous Double Auction

This section introduces the main elements related to the continuous double auction, and shows the details of the *TRADEexpress* implementation which are necessary to understand the developed model.

The order book. The component of a trading system where different trade orders referring to a certain item are stored and matched is called *order book*. The order book is usually represented as a table with two columns called *sides*. These columns respectively contain the buy (*bid*) orders, and the sell (*ask*) orders sorted in descending priority order (see Fig. 2).

Bid orders	Ask orders
b_0 : bid order with highest priority	a_0 : ask order with highest priority
\vdots	\vdots
b_n : bid order with lowest priority	a_m : ask order with lowest priority

Fig. 2. An order book

Trade orders. A trade order is a request to buy or sell a certain quantity of a financial item. A *TRADEexpress* order has the following main attributes:

- Side: bid or ask. The *opposite side* of “ask” refers to “bid” and vice versa.
- Quantity q : amount of units to trade.
- Minimum quantity mq : minimum part of the order quantity that must be matched for the order to be allowed to trade.
- Limit price lp : worst price (highest if the order is a bid, lowest if the order is an ask) that the order can accept in a trade. It can be either fixed by the trader or derived by the system from the state of the order book: *market*

price: the order gets a limit price so that it can match any other order in the order book; *pegged price*: the order gets the same limit price as the best order with a fixed limit price on its side.

- Duration: validity period of the order. Ranges from no limit to a single match opportunity, where the order gets an opportunity to trade and is immediately canceled if it does not succeed.

An order is written as $q (\geq mq) @ lp$, where lp is determined in one of the three ways mentioned above. These attributes can be combined to form the most common order types (listed in Table 1).

Table 1. Common order types and their attribute values

Order type	mq	lp	Duration
Market	0	market price	one match
Limit	0	fixed	trading day
Fill-or-kill	q	fixed	one match
Fill-and-kill	0	fixed	one match
All-or-none	q	fixed	trading day
Pegged	0	pegged price	trading day

To encourage orders that increase the liquidity of the order book, the following prioritization criteria, sorted by precedence, are applied to each side:

1. Limit price: orders with better limit prices (higher for bid orders, lower for ask orders) get higher priority.
2. Minimum quantity: orders without minimum quantity get higher priority.
3. Time of entry: orders entered earlier get higher priority.

Matching mechanism. Continuous double auction is the applied trading mechanism during most of the execution time of *TRADEexpress*. In a continuous double auction, traders are allowed to enter, update, and cancel orders in the order book at any time. Whenever the trading rules allow it, bid and ask orders are matched into trades.

The attributes of an order and its position in the order book determine if the order can be matched against orders on the opposite side. In a valid matching, minimum quantity constraints must always be satisfied, and the limit prices between matched orders must be compatible. Order priority is enforced by allowing an order to match only if all the orders with higher priority and no minimum quantity are matched as well.

Matching in *TRADEexpress* is done in two steps. Every time a trade order is entered or updated, a match attempt is performed, where the incoming order is greedily compared with the orders on the opposite side. For performance reasons, the capacity of the match to find trades is limited. Therefore, if the system detects that there are still potential trades, a second match attempt called *re-match* is performed. In the re-match, all possible combinations between orders from both sides are explored (see Fig. 3).

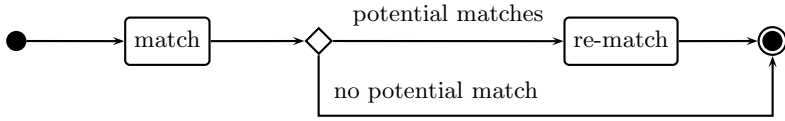


Fig. 3. Two-step matching in the *TRADEexpress*' continuous double auction

In some scenarios, an order with high priority might block potential trades where orders with lower priority are involved. This might happen, for example, if the high-priority order cannot add its quantity to any other order on its side for trading against an all-or-none order. In these cases, a cancellation of the high-priority order gives way for the blocked orders to match. Because of this, *TRADEexpress* executes a re-match after an order cancellation.

3 The Test Oracle

To be able to provide the expected trades after every order action, the test oracle holds a model of the state of the *TRADEexpress* order book. In this model, orders of all types shown in Table 1 are represented by their quantity q , minimum quantity mq , limit price lp , and position in the order book.

Every time an order is entered, updated, or canceled, the order book model is updated accordingly, and the limit price lp of each order is calculated following the rules given in Section 2. Then, a corresponding order matcher problem is generated and solved with the aid of a constraint programming system. The order book model is updated with the results, and the process is repeated for the re-matcher problem. Finally, the combination of calculated trades for both problems is compared with the output generated by *TRADEexpress*. Fig. 4 shows the structure of the test oracle.

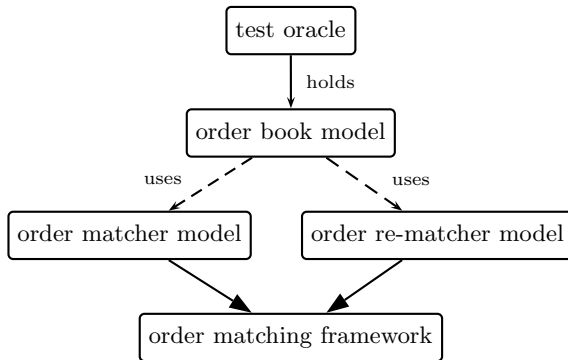


Fig. 4. Structure of the test oracle

3.1 The Order Matching Framework

The order matching framework defines the integer variables that represent the quantity matched between different orders, some auxiliary variables that are useful in the modeling of more advanced rules and constraints that set the basic limitations in how quantities can be distributed.

Input data. An order book with n bid orders and m ask orders sorted by decreasing priority ($n, m \geq 1$):

Bid orders	Ask orders
$b_0 : q_{b_0} (\geq mq_{b_0}) @ lp_{b_0}$	$a_0 : q_{a_0} (\geq mq_{a_0}) @ lp_{a_0}$
\vdots	\vdots
$b_{n-1} : q_{b_{n-1}} (\geq mq_{b_{n-1}}) @ lp_{b_{n-1}}$	$a_{m-1} : q_{a_{m-1}} (\geq mq_{a_{m-1}}) @ lp_{a_{m-1}}$

where q_b, mq_b and lp_b represent the quantity, minimum quantity and limit price of the bid order b . For the sake of simplicity, we define the set of bid orders $B = \{b_i \mid 0 \leq i < n\}$ and the set of ask orders $A = \{a_j \mid 0 \leq j < m\}$.

Variables. The following non-negative integer variables represent how the quantities of the given orders are distributed in a match:

	a_0	\dots	a_{m-1}	total
b_0	tq_{b_0, a_0}	\dots	$tq_{b_0, a_{m-1}}$	tq_{b_0}
\vdots	\vdots	\vdots	\vdots	\vdots
b_{n-1}	tq_{b_{n-1}, a_0}	\dots	$tq_{b_{n-1}, a_{m-1}}$	$tq_{b_{n-1}}$
total	tq_{a_0}	\dots	$tq_{a_{m-1}}$	tq

where $tq_{b,a}$ represents the quantity traded between the bid order b and the ask order a . The auxiliary variables tq_b, tq_a , and tq represent the total traded quantity of the bid order b , the ask order a , and between all orders:

$$tq_b = \sum_{a \in A} tq_{b,a} \quad \forall b \in B \quad \text{and} \quad tq_a = \sum_{b \in B} tq_{b,a} \quad \forall a \in A$$

$$tq = \sum_{b \in B} tq_b = \sum_{a \in A} tq_a$$

Constraints. The framework constraints impose the basic rules on how quantities can be distributed in a generic match problem, considering limit prices, maximum and minimum quantities, and order priorities.

Limit price. Two orders can only match if the limit price of the bid order is greater or equal than the limit price of the ask order:

$$lp_b < lp_a \implies tq_{b,a} = 0 \quad \forall b \in B, a \in A \quad (1)$$

Maximum quantity. Orders cannot trade over their maximum quantities:

$$tq_x \leq q_x \quad \forall x \in B \cup A$$

Minimum quantity. Orders can only trade above their minimum quantities or not at all:

$$tq_x \geq mq_x \vee tq_x = 0 \quad \forall x \in B \cup A \quad (2)$$

Order priority. An order without minimum quantity constraint cannot be bypassed by another order with lower priority:

$$tq_{b_i} < q_{b_i} \wedge mq_{b_i} = 0 \implies tq_{b_j} = 0 \quad \forall b_i, b_j \in B : i < j \quad (3)$$

$$tq_{a_i} < q_{a_i} \wedge mq_{a_i} = 0 \implies tq_{a_j} = 0 \quad \forall a_i, a_j \in A : i < j \quad (4)$$

3.2 The Order Matcher Model

The order matcher model corresponds to the first step in the *TRADEexpress*' continuous double auction. In this problem, the incoming order is sequentially matched against the opposite side, starting by the order with highest priority.

Input data. The input data to the order matcher model is the order book inherited from the matching framework and the incoming order c together with its side $C : c \in C, C \in \{B, A\}$.

Variables. The order matcher uses only the traded quantity variables inherited from the matching framework (as described in the previous subsection).

Objective function. The solution must maximize the total traded quantity:

$$\text{maximize}(tq) \quad (5)$$

Constraints. The order matcher constraints impose that only the incoming order can trade on its side and the matching is performed sequentially.

Incoming order. The incoming order c is the only one that can trade on its side:

$$tq_x = 0 \quad \forall x \in C - \{c\} \quad (6)$$

Sequential matching. An order on the non-incoming side whose quantity, added to the accumulated traded quantity of orders with higher priority, still fits

into the quantity of the incoming order c cannot be bypassed by orders with lower priority:

$$c \in A \wedge \text{atq}(b_i) + q_{b_i} \leq q_c \wedge tq_{b_i} < q_{b_i} \implies tq_{b_j} = 0 \quad \forall b_i, b_j \in B : i < j \quad (7)$$

$$c \in B \wedge \text{atq}(a_i) + q_{a_i} \leq q_c \wedge tq_{a_i} < q_{a_i} \implies tq_{a_j} = 0 \quad \forall a_i, a_j \in A : i < j \quad (8)$$

where $\text{atq}(x_i)$ is a function from orders to integers that represents the accumulated traded quantity from x_0 to x_{i-1} :

$$\begin{aligned} \text{atq}(x_0) &= 0 \\ \text{atq}(x_i) &= \begin{cases} \text{atq}(x_{i-1}) + q_{x_{i-1}} & \text{if } \text{atq}(x_{i-1}) + q_{x_{i-1}} \leq q_c \\ \text{atq}(x_{i-1}) & \text{otherwise} \end{cases} \quad \text{for } i > 0 \end{aligned}$$

Example. Let us consider the following order book, where the incoming order a_0 has received the highest priority on the ask side due to its limit price:

Bid orders	Ask orders
$b_0 : 10 @ 10$	$\rightarrow a_0 : 18 @ 8$
$b_1 : 10 (\geq 10) @ 9$	$a_1 : 30 (\geq 30) @ 9$
$b_2 : 5 (\geq 5) @ 9$	
$b_3 : 3 @ 7$	

The only order allowed to trade on the ask side is a_0 (6). To maximize the total traded quantity (5), b_0 must trade all its quantity. Otherwise, the lower priority orders b_1, b_2 and b_3 would not be allowed to match at all (3). Given that b_0 trades all its quantity, b_1 cannot trade due to its minimum quantity constraint (2). Finally, b_2 contributes to the objective by trading all its quantity. The limit price of b_3 is not compatible with that of a_0 (1). Hence, the expected trades are:

$$b_0 \leftrightarrow a_0 : 10 \qquad b_2 \leftrightarrow a_0 : 5$$

As this example illustrates, the order matcher always finds a single valid solution. This is enforced by the order priority (3, 4) and sequential matching (7, 8) constraints.

3.3 The Order Re-matcher Model

The order re-matcher model corresponds to the second step in *TRADEexpress*' continuous double auction (see Fig. 3). This problem is used as a complement to the order matcher when this, due to its restrictive constraints, is not able to calculate all potential trades in the order book. Although both problems share the same basic structure, their variables and constraints differ. Because of this, it is not possible for the order re-matcher to reuse the solution to the order match problem.

In an order re-match, all possible combinations between order quantities are considered. All orders participating in trades must have a limit price compatible with a reference price, called the *equilibrium price*, which is selected to maximize the objective function. Orders without minimum quantity constraints and better limit price than the equilibrium price are called *must trade* orders, because all their quantity must be matched in a valid solution.

Input data. The input data is the order book inherited from the general matching framework.

Variables. Apart from the traded quantity variables inherited from the matching framework, the following variables are added for re-matching:

- A non-negative integer variable eqp , which represents the equilibrium price of the order matching and whose domain is the set of different order limit prices $\{lp_x \mid x \in B \cup A\}$.
- An integer variable im representing the imbalance in a calculated matching:

$$im = \sum_{\substack{b \in B: \\ lp_b = eqp, mq_b = 0}} (q_b - tq_b) - \sum_{\substack{a \in A: \\ lp_a = eqp, mq_a = 0}} (q_a - tq_a) \quad (9)$$

- Non-negative integer variables tq_0, \dots, tq_{n+m-1} , where tq_k represents the total traded quantity between orders whose indexes sum up to k :

$$tq_k = \sum_{\substack{b_i \in B, a_j \in A: \\ i+j=k}} tq_{b_i, a_j} \quad \forall k : 0 \leq k < n + m$$

The main purpose of these variables is to reflect the total traded quantity in different priority levels. Graphically, each priority level tq_k corresponds to a counter-diagonal in the traded quantity matrix. For example, a re-match problem with $n = m = 3$ has five priority levels. tq_0 corresponds to the darkest diagonal in the following representation:

	a_0	a_1	a_2
b_0	tq_{b_0, a_0}	tq_{b_0, a_1}	tq_{b_0, a_2}
b_1	tq_{b_1, a_0}	tq_{b_1, a_1}	tq_{b_1, a_2}
b_2	tq_{b_2, a_0}	tq_{b_2, a_1}	tq_{b_2, a_2}

Objective function. In contrast to the order matcher problem, for an order re-match problem it is possible to obtain several solutions that maximize the total traded quantity tq . If this happens, it is desirable to get a solution where the unmatched quantity of orders in the equilibrium price level is balanced on both sides of the order book. This can be obtained by minimizing the absolute imbalance $|im|$. If several solutions present the same total traded quantity

and absolute imbalance, trades between orders positioned higher in the order book are prioritized by lexicographically maximizing the different priority levels tq_0, \dots, tq_{n+m-1} .

In summary, the solution must lexicographically maximize the following tuple of integer variables (note that the absolute imbalance $|im|$ is to be minimized):

$$\text{maximize}((tq, -|im|, tq_0, \dots, tq_{n+m-1})) \tag{10}$$

Constraints. The order re-matcher constraints define the effect of the equilibrium price on which orders can trade and impose that the whole quantity from *must trade* orders is always traded.

Equilibrium price. An order that has worse limit price than the equilibrium price eqp cannot trade:

$$\begin{aligned} lp_b < eqp &\implies tq_b = 0 \quad \forall b \in B \\ lp_a > eqp &\implies tq_a = 0 \quad \forall a \in A \end{aligned} \tag{11}$$

Must trade quantity. The quantity from all *must trade* orders on both sides must be completely matched:

$$\sum_{\substack{b \in B: \\ lp_b > eqp, mq_b = 0}} (q_b - tq_b) = 0 \quad \text{and} \quad \sum_{\substack{a \in A: \\ lp_a < eqp, mq_a = 0}} (q_a - tq_a) = 0 \tag{12}$$

Example. Let us consider the following order book:

Bid orders	Ask orders
$b_0 : 10 (\geq 10) @ 19$	$a_0 : 6 (\geq 6) @ 12$
	$a_1 : 9 (\geq 9) @ 12$
	$a_2 : 5 @ 16$
	$a_3 : 5 (\geq 5) @ 17$

There are several possible pairs of ask orders that can trade the total quantity of b_0 , maximizing the total traded quantity (10):

Pair	eqp	$ im $
$\{a_0, a_2\}$	16	1
$\{a_1, a_2\}$	16	4
$\{a_2, a_3\}$	17	0

If one of the two first pairs is chosen, the equilibrium price is set to 16, so that a_2 can trade (11). In that case $lp_{a_2} = eqp$, and a_2 adds its unmatched quantity to the imbalance (9). If the pair $\{a_2, a_3\}$ is chosen, the equilibrium price becomes 17, and a_2 becomes a *must trade* order, no longer contributing to the imbalance.

Because the must trade constraint (I2) is satisfied, and the absolute imbalance $|im|$ is minimized (I0), the pair $\{a_2, a_3\}$ is chosen to trade with b_0 :

$$b_0 \leftrightarrow a_2 : 5 \qquad b_0 \leftrightarrow a_3 : 5$$

3.4 Why Constraint Programming?

As mentioned in the introduction, accuracy and low development cost are two essential goals in the design of test oracles. These goals require that the implementation of the test oracle is kept simple and easily traceable to the specification. The main reason for using constraint programming has been therefore the ease of implementing every concept from the matcher and re-matcher models, including logical expressions (3) and lexicographic optimization (I0).

In particular, the chosen system (Gecode) has allowed us to express the lexicographical optimization straightforwardly, by adding new lexicographic constraints every time a better solution is found. Another relevant factor in the decision to use constraint programming has been the availability of solvers distributed as object-oriented libraries. This feature has contributed to a simpler and thus less error-prone interface with Cinnober's JUnit-based test framework.

4 Results

We evaluate the constraint programming approach to the test oracle problem from two angles. First, we show the failures detected by the oracle in the execution of random tests against *TRADEexpress*. Then, we complete the analysis by giving results related to the design goals of the test oracle.

The constraint models used by the test oracle have been implemented in the constraint programming system Gecode (I1), version 3.2.0. The branching strategy for all variables in both models has been to branch on the variable with the smallest domain. The values are selected by splitting the domain into two subsets, exploring the larger subset first. The search has been executed on a single core. All tests and measurements have been performed on a Linux machine with a Quad-Core Intel Xeon 2.5 GHz processor and 4 GB of main memory.

Random tests. We have considered a scenario designed to exploit all capabilities of the test oracle. It includes three traders entering, updating, and canceling random orders of different types. Table 2 shows the designed configuration, with the probabilities of traders and actions being chosen in each cycle.

We have run 500 test cases with different random generator seeds. Each test case comprises 100 order actions each, with prices generated in a range between 10 and 100, and quantities generated in a range between 2 and 50.

Although we planned a total of 50 000 order actions, premature terminations in several test cases due to the detection of failures have reduced the total order actions to 45 144. Table 3 shows the total match and re-match calculations with positive traded quantity (called *hits*) and their corresponding expected trades.

Table 2. Random traders, actions and probabilities in the executed test cases

Trader	Probability	Action	Probability
A	30%	Enter a limit order	80%
		Update an order	10%
		Cancel an order	10%
B	30%	Enter a marker order	33.3%
		Enter a fill-or-kill order	33.3%
		Enter a fill-and-kill order	33.3%
C	40%	Enter an all-or-none order	40%
		Enter a pegged order	40%
		Update an order	10%
		Cancel an order	10%

Table 3. Total match and re-match hits and expected trades in the test runs

Matching step	Total hits	Total trades
Match	10 895	17 037
Re-match	526	1189

Table 4. MTBF of the failures detected in the random test cases

Failure	Occurrences	MTBF (order actions)
Pegged order in empty order book	51	885
Incoming low priority order matches	37	1220
Total	88	513

After applying manual analysis to the failing test cases, we were able to identify two types of failures in *TRADEexpress* reported by the test oracle. In order to better understand the corresponding risks, we have calculated the individual and total Mean Time Between Failures (MTBF) during the execution of the random tests. The results, expressed in number of order actions, are shown in Table 4. Further detail about each type of failure can be found in (12, Ch. 6):

A pegged order remains in an empty order book. An invariant of the order book in *TRADEexpress* is that a side that does not contain any limit order cannot contain a pegged order, because pegged orders require limit orders to base their price on. This invariant is verified by the test oracle at the end of the two-matching step calculations, and was violated in several test scenarios.

An incoming order without best priority is allowed to match. In some specific situations, constraints 3 and 4 in Section 3.2 were violated by *TRADEexpress*, allowing orders without the best priority to match when higher-priority orders did not.

These two failures correspond to bugs in the core of the *TRADEexpress* matching logic. Both bugs have been reported and fixed by the time of writing this paper. These bugs had been most likely present in the system for several years. However, the infrequent test cases needed to reproduce them had not been generated in more systematic testing approaches. Previous random testing approaches were not able to detect them either, because of the lack of a complete test oracle [3, Ch. 6]. The detection of these failures can be seen, thus, as empirical evidence of the effectiveness of the constraint programming approach to the test oracle problem.

Furthermore, in the requirements formalization process for formulating the constraint model, we have detected six documentation defects, classified into two categories: ambiguous statements due to the use of a natural language (English); erroneous calculations in illustrative examples.

Main goals of the test oracle. As mentioned in the introduction, the main goals for the design of the oracle have been completeness, correctness, accuracy, and low development cost. Performance has been a secondary goal, because the execution of the random tests does not require manual intervention and is not subject to time pressure.

Completeness. Table 1 shows the main order types supported by *TRADEexpress*. A more complete list is given in (13). The order types *Good till Date*, *Good for Day*, *Good till Canceled*, and *Good until Next Uncross* are special cases of the limit order, differing only in their duration, and are thus covered by the test oracle.

The test oracle covers 10 out of the 13 order types listed in (13), that is, a 77% of the order types supported by *TRADEexpress*. Extending the model to support the remaining *Stop-loss*, *Iceberg* and *Dark* order types is left as future work.

Accuracy and correctness. The use of a declarative programming paradigm such as constraint programming reduces the proof of correctness of the continuous double auction model to verifying the specification itself. Furthermore, it makes the oracle more independent from the system under test, which is developed following an imperative paradigm. This independence contributes to the effectiveness of the test oracle, as it reduces the likelihood of sharing design and implementation bugs with the system under test [4].

Development cost. The development of the test oracle, including modeling, implementation and testing, took approximately 200 person-hours. All used technologies, including the constraint programming system, are freely available. We have estimated the maintainability by counting lines of code (14). The test oracle has 1946 lines of code, approximately only 20% of the code that implements the modeled functionality in *TRADEexpress*. Furthermore, the declarative nature of the model makes it easy to trace changes in the system requirements, which contributes to lower maintenance costs.

Performance. We have measured the average execution time taken by Gecode to solve the different order match and re-match problems generated during the execution of 60 test cases based on the scenario shown in Table 2. Due to the limited accuracy of the time measurement functions in the considered orders of magnitude, the average time of 50 executions has been taken for each match and re-match problem.

In a total of 4256 instances, the order match problem is solved in 161 μ s on average, with a coefficient of deviation of 85%. The order re-match problem as a more complex combinatorial problem takes 388 μ s on average, with a coefficient of deviation of 61%, for a total of 3534 instances. Some degenerate re-match problems designed to stress-test the oracle take up to 155 ms to be solved, which is still an order of magnitude more efficient than our initial design goal.

5 Conclusion and Future Work

This paper has introduced a constraint-based test oracle for a continuous double auction from a real-life trading system. It has shown that constraint programming meets the particular goals for the design of test oracles, providing accuracy and ease of modeling in a cost-effective way. The significance of this approach is witnessed by finding actual, relevant bugs in a widely-deployed, thoroughly tested trading system. As a side benefit, several defects in the system requirements have been found.

The results obtained in this paper support the importance of using formal models in the development of complex financial systems such as *TRADEexpress*. Constraint programming adds the ability to make these models executable with low development cost. This application improves significantly the effectiveness of the testing process, and has raised interest at Cinnober: the company plans to continue the formalization process initiated by this research and to use the test oracle for regression purposes.

An obvious way to improve the effectiveness of the test oracle is to extend the continuous double auction model to cover additional order types supported by *TRADEexpress*, as suggested in Sect. 4. Considering a more global perspective, the emergence of open protocols such as the Financial Information eXchange protocol (FIX) [15] opens an opportunity to extend the application presented in this paper to a variety of trading systems. Future work in this direction would include the development of a constraint-based test oracle implementing a relevant subset of a widely adopted protocol such as FIX.

Acknowledgments. The authors are grateful for helpful comments from Mikael Z. Lagerkvist, Carles Tomás Martí and the anonymous reviewers.

References

- [1] Wagner, W.H.: Electronic trading: rival or replacement for traditional floor-based exchanges? In: World of Exchanges: Adapting to a New Environment. Euromoney Books (2007)

- [2] Friedman, D., Rust, J. (eds.): The double auction market: institutions, theories, and evidence. Addison-Wesley, Reading (1993)
- [3] Höjeborg, N.: Random tests in a trading system: random tests in a trading system using simulations and a test oracle. Master's thesis, School of Computer Science and Communication. KTH Royal Institute of Technology, Sweden (2008)
- [4] Hoffman, D.: Using oracles in test automation. In: Proceedings of Pacific Northwest Software Quality Conference, pp. 90–117 (2001)
- [5] Gotlieb, A., Botella, B., Rueher, M.: Automatic test data generation using constraint solving techniques. SIGSOFT Softw. Eng. Notes 23(2), 53–62 (1998)
- [6] Meudec, C.: ATGen: automatic test data generation using constraint logic programming and symbolic execution. Software Testing Verification and Reliability 11(2), 81–96 (2001)
- [7] Bin, E., Emek, R., Shurek, G., Ziv, A.: Using a constraint satisfaction formulation and solution techniques for random test program generation. IBM Systems Journal 41(3), 386–402 (2002)
- [8] Ryu, Y.U.: Hierarchical constraint satisfaction of multilateral trade matching in commodity auction markets. Annals of Operations Research 71, 317–334 (1997)
- [9] Kalagnanam, J.R., Davenport, A.J., Lee, H.S.: Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. Electronic Commerce Research 1(3), 221–238 (2001)
- [10] Hasbrouck, J.: Empirical Market Microstructure: The Institutions, Economics, and Econometrics of Securities Trading. Oxford University Press, Oxford (2007)
- [11] Gecode Team: Gecode: Generic constraint development environment (2006), <http://www.gecode.org>
- [12] Castañeda Lozano, R.: Constraint programming for random testing of a trading system. Master's thesis, School of Information and Communication Technology. KTH Royal Institute of Technology, Sweden (2010)
- [13] Cinnober Financial Technology AB: TRADExpress Trading System product sheet (2010), http://www.cinnober.com/files/TS_ProductSheet_0.pdf
- [14] Zuse, H.: A Framework of Software Measurement. Walter de Gruyter & Co., Hawthorne (1997)
- [15] FIX Protocol Limited: Financial Information eXchange Protocol (2010), www.fixprotocol.org

A Safe and Flexible CP-Based Approach for Velocity Tuning Problems

Michaël Soullignac¹, Michel Rueher², and Patrick Taillibert³

¹ ISEN Lille, 41 Boulevard Vauban, 59046 Lille Cedex, France
`michael.soullignac@isen.fr`

² Nice Sophia Antipolis University, I3S/CNRS, BP 145, France
`michel.rueher@gmail.com`

³ THALES Aerospace, 2 Avenue Gay Lussac, 78852 Elancourt, France
`patrick.taillibert@fr.thalesgroup.com`

Abstract. This paper introduces a new velocity tuning approach for autonomous vehicles based on Constraint Programming (CP) over continuous domains. We use CP to compute a safe approximation of configurations where collisions with obstacles may occur or technological limits may be violated. The use of CP leads to a flexible approach, facilitating the incorporation of new characteristics, e.g., constraints modeling the influence of currents. We illustrate these capabilities offered by CP in the context of UAV missions. Experimental results obtained on actual wind charts are provided.

1 Introduction

Recent advances made in the field of autonomous vehicles suggest that, in a near future, Unmanned Air Vehicles (UAVs) or Autonomous Underwater Vehicles (AUVs) will be more and more deployed in order to achieve various missions such as surveillance, intelligence or search and rescue. For physical or strategic reasons, these vehicles may not be able to receive directly orders from a headquarter in real-time. Thus, they have to embed their own motion planner. Because the environment is often changing or unknown, this planner has to be very reactive.

Motion planning among mobile obstacles is commonly divided into two tasks: path planning and velocity tuning. Numerous algorithms have been proposed for specific instances of velocity tuning. The main drawback of these approaches is that minor changes in the characteristics of the application may require deep changes in the algorithms.

That is why we propose a flexible approach based on Constraint Programming (CP) techniques. Practically, we use CP on continuous domains to identify the unreachable regions of a 2D space-time, modeling potential collisions with obstacles and/or violations of the technological limits of the vehicle. Thus, we can ensure that the deduced time-minimal trajectory is collision-free and feasible. An essential contribution of CP comes from the fact that it allows to easily incorporate new constraints. We illustrate this point by showing that constraints

modeling the presence of currents can be added to the initial constraint system with a limited rework effort. First experimental results obtained on real wind charts are encouraging.

2 The Problem

2.1 Informal Description

A punctual vehicle, starting from a site A , has to compute a time-optimal trajectory to a goal site B , in a planar environment containing static and mobile obstacles, as illustrated in figure 1. This computation is done in two steps.

First, a path planning step, computing a path \mathcal{P} avoiding static obstacles. Any path planning method from literature can be used in this step. The only assumption about the path \mathcal{P} is that it is made up of successive line segments.

Second, a velocity tuning step, minimizing the arrival time at B and avoiding mobile obstacles, with a bounded velocity. This second problem is addressed in this paper.

2.2 Formalization

The environment is modeled by a 2-D Euclidean space E , with a frame of reference $R = (0, x, y)$. In R , the coordinates of a vector \vec{u} are denoted (u_x, u_y) and its magnitude u . In particular, the vehicle's velocity vector relative to the frame R (ground speed) is denoted \vec{v} , and its magnitude v .

The path \mathcal{P} is defined by a list V of n viapoints, denoted V_i . Each viapoint V_i is located on \mathcal{P} at curvilinear abscissa l_i . Two successive viapoints (V_{i-1} and V_i) are linked by a line segment. In other terms, \mathcal{P} is made up of successive line segments.

Each mobile obstacle O_i is modeled by a rectangle of size $S_{i_x} \times S_{i_y}$, performing successive straight line moves at constant velocity (dashed lines in fig. 1). This rectangle corresponds to a punctual mobile surrounded by a rectangular safety zone. Note that this safety zone can be easily extended to a polygon.

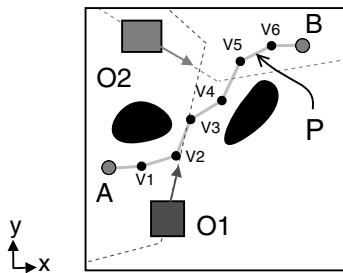


Fig. 1. A velocity tuning problem with 2 mobile obstacles (grey rectangles). The pre-computed path \mathcal{P} already avoids static obstacles (dark regions).

Our problem consists in finding a timing function $\sigma : M \in \mathcal{P} \mapsto t \in [0, T]$ minimizing the arrival time $t_B = \sigma(B)$, with respect to the following constraints: (1) maximal velocity: $v \leq v_{max}$, (2) mobile obstacles avoidance and (3) the latest arrival time T to B , due for instance to the embedded energy or visibility conditions. In other terms, our problem can be seen as mapping the path \mathcal{P} to a trajectory.

3 Why CP?

The existing velocity tuning approaches generally work in a 2-D space-time. The first dimension $l \in [0, L]$ (where L is the length of the path) is the curvilinear abscissa on the path. The second one, $t \in [0, T]$ (where T is the latest arrival time), is the elapsed time since departure. As illustrated in figure 2, each point of the path is represented by a vertical line in the space-time. In particular, start and goal sites are represented by the extreme left and right vertical lines. Moreover, each mobile obstacle O_i generates a set of *collision surfaces* S_j in the space-time. These surfaces contain all pairs (l, t) leading to a collision between the vehicle and O_i .

Once the space-time is built, the initial velocity tuning problem can be reformulated into a path planning problem in this space-time. However, this space-time has specific constraints, notably due to time monotony or velocity bounds. Therefore, specific methods have been developed [10][20].

These methods are based on a discretization of the environment into elementary entities: line segments (fig. 3a) or polygonal cells (fig. 3b). These entities are then modeled as nodes of a graph.

The initial -concrete- path planning problem is thus reformulated into an abstract one: finding the shortest path in a graph. This problem is generally solved by applying an adaptation of a classical search algorithm, such as A^* [6] or one of its numerous variants.

As we can see, in the above approaches, unreachable regions of the space-time only materialize collisions with mobile obstacles. Other constraints about the vehicle and the environment are implicit, and are taken into account in the search

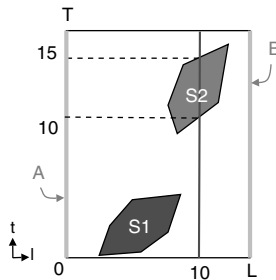


Fig. 2. The space-time corresponding to the example of figure 1, containing two collision surfaces S_1 and S_2 . A collision necessarily occurs at abscissa $l = 10$ between $t = 10$ and $t = 15$.

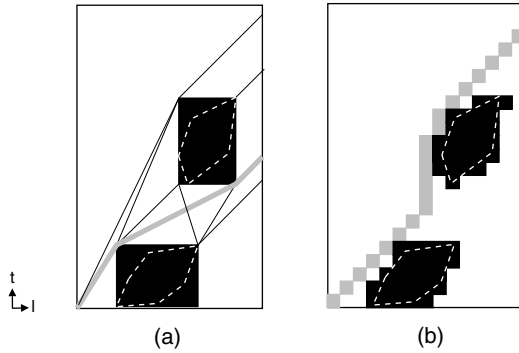


Fig. 3. (a) path (in light grey) found using the visibility graph method [10]; (b) path found using the cell decomposition method [20]

algorithm. This has two major drawbacks: (1) the space-time does not reflect all the constraints of the problem, in particular the technological limits of the vehicle and (2) the search algorithm is customized to exploit some particular properties of the considered instance. Therefore, if the properties of this instance evolve, significant changes in the design and the implementation of this algorithm may be required.

The ability to add new constraints in a declarative way is the major advantage of using CP.

4 How CP?

As said before, we use CP on continuous domains to compute a safe approximation of unreachable regions in the space-time. This is done in two steps:

The first step consists in computing the *collision surfaces* described in section 3, that is regions in the space-time materializing collisions between obstacles and the vehicle. This step is done by solving a set of local CSPs, called *ColSurf_{ij}*.

The second step consists in aggregating unreachable regions of the space-time, due to mobile obstacles (collision surfaces) and those due to the technological limits of the vehicle. This step is done by solving a global CSP, called *UnreachReg*. We will show that this CSP can be easily extended to include new constraints about the vehicle or the environment. We will illustrate this capability by adding currents in the initial problem.

All CSPs are defined on continuous domains. This choice is first explained, and next the two steps mentioned above are presented.

4.1 Modeling Space-Time Regions with Constraints over Continuous Domains

Modeling space-time regions in velocity-tuning problems can be done either with constraints over finite domains or over continuous domains. As explained in our

previous paper [18], discrete domains impose to discretize quantities which are continuous by nature in the tuning problem, such as time. This leads to two issues:

1. If both time horizon T and the required precision on time τ are high, domains for variables modeling time may become huge. For instance, if $T = 10h$ and $\tau = 1s$, each domain contains 36000 values.
2. Discretizing time may lead to "miss" some collisions. Indeed, if no collision is detected at time t_{i-1} , and no collision is detected at $t_i = t_{i-1} + \tau$, it is generally assumed that no collision can occur between t_{i-1} and t_i . In fact, this assumption becomes more and more probable when τ tends towards 0, but it is never guaranteed.

To avoid these issues, we choose to model 2D space-time regions with constraints over continuous domains. The weak point of constraints over continuous domains is that they cannot in general guarantee that a solution exists in a box, even if this box is very small. The strong property of constraints over continuous domains is their refutation capabilities.

For this reason, the CSP described below is not used to model reachable regions but to describe an over-approximation of unreachable regions.

Indeed, since unreachable regions are over-estimated the remaining regions are necessarily reachable. However, as explained later, our approach is incomplete, because this over-estimation may eliminate the solutions of some very constrained problems.

4.2 Preliminaries: Computing Collision Surfaces in the Space-Time

In a CSP that we call *ColSurf_{ij}* (for "Collision Surface", pair of moves (i, j)), we model the following situation: (1) the vehicle and a mobile obstacle are moving on line segments i and j of their respective trajectory, and (2) a collision occurs between them. Solving this CSP may lead to two results:

1. The CSP is inconsistent, then we are sure that no collision will occur during this pair of moves.
2. The CSP is consistent, some collisions may occur in a time interval denoted D_{ij}^c .

Therefore, if the interval D_{ij}^c is not empty, it represents a part of a collision surface. By repeating the process described above for all possible values of i and j , we are able to compute an outer approximation of collisions surfaces.

1) Formulation of the CSP *LocalCol_{ij}*

Let us consider that the vehicle and a mobile obstacle are moving line segments i and j of their respective trajectory:

- the vehicle is moving between viapoints $V_{i-1} = (x_{i-1}, y_{i-1})$ and $V_i = (x_i, y_i)$,

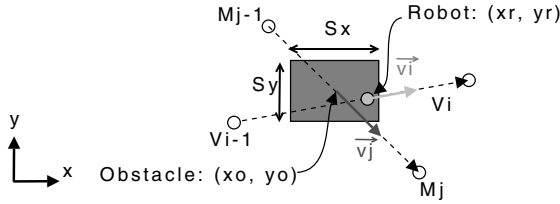


Fig. 4. A collision between the vehicle and a mobile obstacle: the vehicle lies to the rectangular safety zone (dark grey)

- an obstacle, of size $S_x \times S_y$ is performing its j th straight line move, at velocity v_j , between two points $M_{j-1} = (x_{j-1}, y_{j-1})$ and $M_j = (x_j, y_j)$, starting at time t_{j-1} .

As depicted in figure 4, a collision between the vehicle and the obstacle occurs if the vehicle lies to the rectangular safety zone.

If we denote (x_r, y_r) the position of the vehicle, (x_o, y_o) the position of the obstacle, and t_c the collision time between them, this situation can be modeled by the following equations:

$$\begin{cases} a_i = y_{i-1} - y_i \\ b_i = x_i - x_{i-1} \\ c_i = y_i \cdot x_{i-1} - x_i \cdot y_{i-1} \\ a_i \cdot x_r + b_i \cdot y_r + c_i = 0 \\ (x_r - x_{i-1}) \cdot (x_r - x_i) \leq 0 \\ (y_r - y_{i-1}) \cdot (y_r - y_i) \leq 0 \end{cases} \quad (VehicleMotion)$$

[The vehicle is moving on the line segment $[V_{i-1}, V_i]$, of equation $a_i \cdot x_r + b_i \cdot y_r + c_i = 0$]

$$\begin{cases} d_{j_x} = x_j - x_{j-1} \\ d_{j_y} = y_j - y_{j-1} \\ d_j = \sqrt{d_{j_x}^2 + d_{j_y}^2} \\ v_{j_x} = v_j \cdot (d_{j_x}/d_j) \\ v_{j_y} = v_j \cdot (d_{j_y}/d_j) \\ x_o = v_{j_x} \cdot (t_c - t_{j-1}) + x_{j-1} \\ y_o = v_{j_y} \cdot (t_c - t_{j-1}) + y_{j-1} \end{cases} \quad (ObstMotion)$$

[The obstacle is moving on the line segment $[M_{j-1}, M_j]$ at velocity v_j , starting at time t_{j-1} . The traveled distance is denoted d_j]

$$\begin{cases} x_r \geq x_o + S_x/2 \\ x_r \leq x_o - S_x/2 \\ y_r \geq y_o + S_y/2 \\ y_r \leq y_o - S_y/2 \end{cases} \quad (Collision)$$

[The vehicle is within the rectangular safety zone of the obstacle, of size $S_x \times S_y$]

In the above equations, the domain of variable t_c represents the set of times where vehicle may collide the obstacle. This domain is used later to approximate collision surfaces.

Of course, in this very simple case –linear moves for the vehicle and obstacles– the domain could be computed analytically or symbolically. However, symbolical computation may become costly as soon as the constraints are more complex (e.g, 3D spaces). On top of it, our goal is to offer a great flexibility, and thus, we do not want to use any *ad hoc* technique.

For instance, consider that the vehicle has to perform a set of circles, modeling a surveillance task. This could easily be done by replacing line equations by circle equations in the above model whereas the analytic resolution would require much more work.

2) Computing collision surfaces

Let us define the interval of *collision times*, denoted T_i^{col} . This interval has the following meaning: if $t_i \in T_i^{col}$, then the vehicle collides a mobile obstacle between V_{i-1} and V_i . Using the CSP described above, each interval T_i^{col} can be computed by the following procedure:

For each mobile obstacle O_k do:

1. Let L_k denote the number of line segments performed by O_k
2. For j from 1 to L_k :
 - (a) Initialize T_i^{col} by : $T_i^{col} \leftarrow \emptyset$
 - (b) Solve the CSP *LocalCol.i.j*
 - (c) If *LocalCol.i.j* is consistent
 - i. Let D_{ij}^c denote the domain of variable t_c
 - ii. Update T_i^{col} by: $T_i^{col} \leftarrow T_i^{col} \cup D_{ij}^c$

In the general case, the interval T_i^{col} computed by this procedure is an union of subintervals, i.e. $T_i^{col} = \cup_{j=1}^{s_i} [t_j^-, t_j^+]$, where s_i is the number of possible collisions between V_{i-1} and V_i .

As shown in figure 5, the whole T_i^{col} models an outer approximation of collision surfaces introduced in section 3. Thus, all possible collisions are captured by T_i^{col} .

That is, contrary to most existing approaches, even if forbidden surfaces are approximated, our approach remains correct: the behavior of the vehicle is guaranteed to be safe. On the other hand, our approach is incomplete, because a too rough approximation can obstruct the space-time. In those conditions, the problem could be considered as unsolvable, even if solutions exist.

4.3 Computing the Unreachable Regions of the Space Time

We define a CSP, called *UnreachReg* (for "Unreachable Regions"), which aggregates the collision surfaces computed in the last step with the unreachable regions due to the technological limits of the vehicle. Here, the only limits taken into account are velocity bounds.

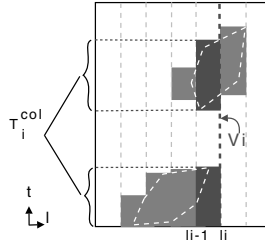


Fig. 5. Collision times T_i^{col} in the space-time of fig. 2, discretizing collision surfaces by a set of bands

Note that only the principle of the CSP is explained below. A detailed description of constraints can be found in the chapter 7 of [16].

1) *Unreachable regions due to excessive velocity*

Let us consider a move of the vehicle on an arbitrary line segment $[V_{i-1}, V_i]$. If we denote respectively v_i , t_i and d_i the vehicle’s velocity, the travel time and the traveled distance on this line segment, these variables are linked by $t_i = t_{i-1} + d_i/v_i$.

Using this relationship, we can model a *too early* arrival at viapoint V_i , that is, an arrival violating the maximal velocity constraint. Indeed, all times t_i verifying this relationship with $v_i > v_{max}$, are, by definition, forbidden. The corresponding constraints generate a "floor" in the space-time, as illustrated in figure 6a.

The same reasoning can be done for too late arrival, with regard to the latest arrival time T . The corresponding constraints generate a "ceiling" in the space-time, as illustrated in figure 6b.

2) *Merging unreachable areas*

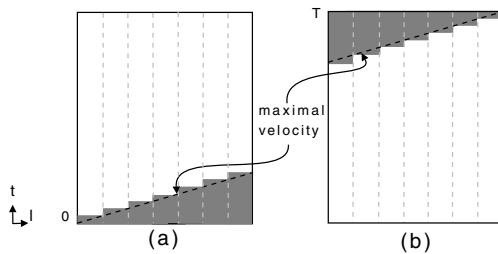


Fig. 6. (a) floor modeling a too early arrival; (b) ceiling modeling a too late arrival. Ceiling and floor are delimited by the dashed line of slope v_{max} .

Unreachable areas due to the mobile obstacles and the technological limits of the vehicle are interconnected. For instance, we have to respect to velocity

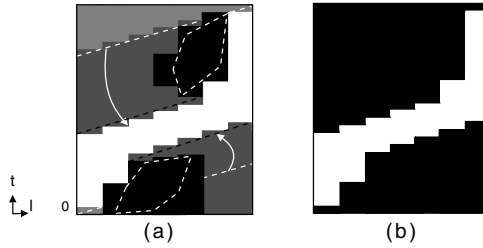


Fig. 7. (a) Shadows (dark grey) obtained by temporally shifting light grey regions (due to velocity bounds); (b) the final space-time, containing extended ceiling and floor

bounds, if we want to speed up the vehicle so that it will arrive at a given point before some obstacle.

In the space-time, collision surfaces will lead to a temporal shifting of the ceiling and the floor of figure 6, as depicted in figure 7a.

Let us explain the shifting of the floor.

The presence of mobile obstacles extends the concept of a too early arrival. Indeed, if an obstacle O_j occupies the viapoint V_i during $D_j = [t_j^-, t_j^+]$, then all arrival time lying in D_j have to be shifted to t_j^+ . In other terms, if the vehicle reach V_i at time $t_i \in D_j$, then it will have to wait until time t_j^+ (that is, until V_i is obstacle-free).

The same reasoning can be done for the ceiling.

The corresponding constraints generate extended ceiling and floor in the space-time (see figure 7b). These unreachable areas merge the effect of moving obstacles and technological limits of the vehicle. Graphically, they can be interpreted as "shadows" behind collision surfaces.

4.4 Deducing the Time-Minimal Path

Once the CSP is numerically solved, the final space-time, containing the extended floor and ceiling is known. Two situations may arise:

1. A corridor exists between the floor and the ceiling (case of fig. 7b). In this case, it is very simple to deduce the time-minimal path in this space-time. For instance, it can be obtained by linking the upper edges of the floor, as depicted in figure 8.
2. The space-time is obstructed (the floor intersects the ceiling). In this case the problem is *a priori* unsolvable, but we cannot be sure about that. However, it is possible that unreachable regions are too roughly approximated.

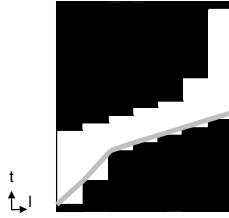


Fig. 8. The time-minimal path (in grey) obtained in the space-time of fig. 7b

4.5 Implementation

The approach introduced in this paper has been implemented in our trajectory planner, *Airplan*, first described in [17].

In *Airplan*, the initial velocity tuning module, based on the *clpfd* solver [5] (discrete domains + arc-consistency), has been replaced by another one, using the *Interlog* solver [4] (continuous domains + 2B-consistency). This new version of *Airplan* will be demonstrated during the conference.

We choose to use a weak local consistency like 2B-consistency [12], because it is light to implement and computationally efficient in our case (see experimental results). Of course, stronger consistencies like *Box* [7], *HC4* / *HC4-Revise* [3], *Quad* [11] or *I-CSE* [2] would probably provide sharper approximations but their implementation is much more costly.

5 Added Value of CP

We showed that a first added value of CP on continuous domains was the correctness: since non-solution spaces may be over-estimated, the computed trajectory is guaranteed to be safe, that is, collision-free.

In this section, we illustrate another benefit of CP: flexibility. In section 4, we presented a global CSP modeling the "core" velocity tuning problem introduced in section 2. Specific constraints have to be added to this CSP to take into account some vehicle's specificities, such as curvature or acceleration constraints, sensors capacities or configuration, etc.

We focus here on the main specificity of Unmanned Aerial Vehicles (UAVs): their sensibility to currents. This can be easily modeled by adding some constraints in the core model.

5.1 Constraints Formulation

In presence of currents, constraints related to the vehicle velocity have to model two additional properties:

- velocity composition law: the velocity of the vehicle depends both on the engine command and on the velocity of the current.

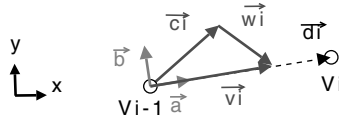


Fig. 9. The velocity composition law between V_{i-1} and V_i

- path following: the vehicle has to compensate the disturbances of the currents to stay on the path. Specific methods have been developed to address this problem [14], but here, the flexibility of CP avoids to use them.

To illustrate this point, let us consider the straight line move \vec{d}_i , between the viapoints V_{i-1} and V_i . For this move, we define: (1) \vec{c}_i the average velocity of the current, (2) \vec{v}_i the vehicle’s velocity relative to the frame R_i , and (3) \vec{w}_i the vehicle’s velocity relative to \vec{c}_i .

As shown in figure 9, velocities \vec{v}_i and \vec{w}_i are linked by the composition law $\vec{v}_i = \vec{w}_i + \vec{c}_i$.

Moreover, since we want to impose the vehicle to stay on the path, \vec{v}_i and \vec{d}_i are collinear.

Using the local frame $R_i = (V_{i-1}, \vec{a}, \vec{b})$, with $\vec{a} = \vec{d}_i/d_i$ and $\vec{a} \perp \vec{b}$, unreachable regions due to an excessive velocity (i.e. greater than ν_{max}) can be expressed as follows in presence of currents:

$$\begin{cases} v_i = c_{ia} + w_{ia} \\ 0 = c_{ib} + w_{ib} \end{cases} \quad (VehicleVel)$$

[velocity composition law in the frame R_i]

$$\begin{cases} c_a = c_i \cdot \cos \alpha \\ c_b = c_i \cdot \sin \alpha \\ \cos \alpha = \vec{c}_i \cdot \vec{d}_i / c_i \cdot d_i = (c_{ia}d_{ia} + c_{ib}d_{ib}) / c_i \end{cases} \quad (CurrentVel)$$

[coordinates of \vec{c}_i in R_i]

$$\begin{cases} w_i^2 = w_{ia}^2 + w_{ib}^2 \\ w_i > \nu_{max} \end{cases} \quad (VelBounds)$$

[maximal velocity relative to the current c_i]

To integrate the influence of currents on the vehicle, the unique constraint $v > \nu_{max}$ introduced in the problem statement has simply to be replaced by the sets of equations (*VehicleVel*), (*CurrentVel*) and (*VelBounds*) described above.

5.2 Result

Figure 10 shows the impact of currents on the previous space-time, and on the solution.

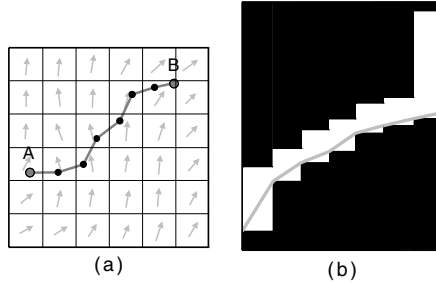


Fig. 10. (a) Adding currents in the initial problem (grey arrow = velocity vectors). (b) Corresponding space-time and solution.

6 Experimental Results

The key idea of this paper is that velocity tuning based on CP leads to a safe and flexible approach. To evaluate the capabilities of this approach, we have performed experiments on actual wind chart for UAV missions.

6.1 The Test-Cases

As shown in figure 11, a test-case is a simple UAV mission. The UAV takes off at A, and then evolves on a precomputed path \mathcal{P} until reaching B. The UAV has to optimize its velocity on \mathcal{P} , to take the wind and mobile obstacles into account.

Each test-case has been generated as follows:

- Wind charts were daily collected during three months on the Meteo France website [13], leading to 100 different realistic environments with currents.
- A (start) and B (goal) points: randomly placed on the wind chart.
- Path: computed using the sliding wavefront expansion, described in [19]. This algorithm provided feasible and smooth paths, even in presence of strong currents.
- Mobile obstacles: each mobile obstacle goes across the environment by performing a straight line move $P_1 \rightarrow P_2$ at constant velocity. P_1 and P_2 are randomly chosen on two borders of the environment, until an intersection I between the path \mathcal{P} and the line segment $[P_1, P_2]$ is detected. Their velocity and size are randomly set in realistic intervals of values.

Note that, in the particular context of UAV missions, both the fact that mobiles obstacles' trajectories are piecewise linear and known in advance are realistic. These assumptions model well flight plans of potential airplanes in the area (possibly updated in real-time from the headquarter) or of other autonomous vehicles, in other to perform a synchronized mission.

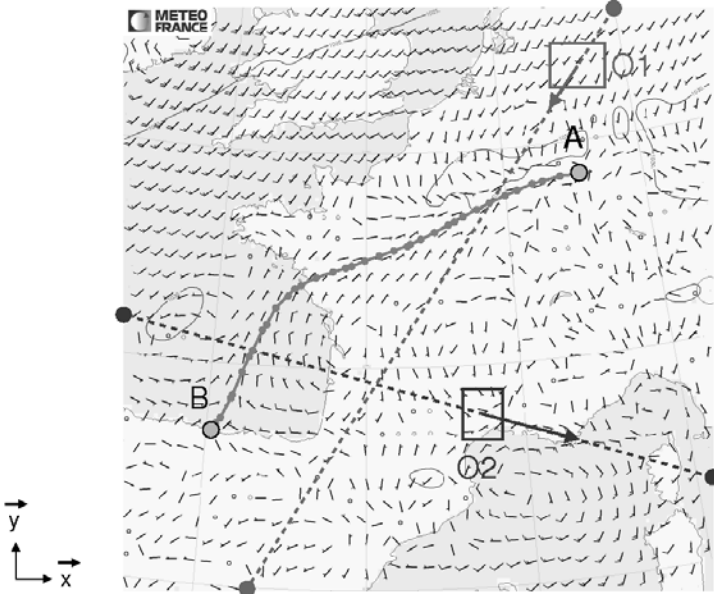


Fig. 11. A test-case with $m = 2$ mobile obstacles. The path is made up of $n = 36$ viapoints, which leads to a CSP containing about 400 variables and 650 constraints.

6.2 Computation Time

The resulting computation times are plotted in figure 12. Each dot is the mean time obtained on 500 test-cases. These results have been obtained by running the last version of Airplan on a 1.7Ghz PC with 1Go of RAM. Note that better results could be obtained by using state of art software such as, for instance: RealPaver [15], ALIAS [1], Icos [9] or Ibex [8].

From a strictly qualitative point of view, the global computation time remains reasonable (under 1 second) even in presence of many moving obstacles. Therefore, our approach is both flexible and computationally efficient.

From a statistical point of view, the computation time appears to be linear. A linear regression confirms this assumption in the range of our tests (with a coefficient of determination $R^2 > 0.999$).

This seems natural, because the size of our CSP is in $O(n \cdot m)$, where n is the number of viapoints on the vehicle's path \mathcal{P} , and m the number of obstacles. Therefore, it was expected that the average computation time was in $O(\tilde{n} \cdot m)$,

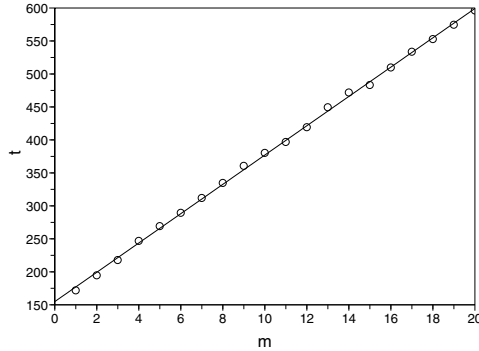


Fig. 12. Computation time t (in ms) in presence of m moving obstacles (white dots: experimental results; black line: linear regression)

where \tilde{n} is the average number of points on \mathcal{P} ($\tilde{n} \approx 20$ in our tests). However, a deeper study is required to confirm these results in the general case.

7 Conclusion

In this paper, we proposed a constraint-based flexible approach for handling velocity tuning problems. CP allows to model the "core" velocity tuning problem, and then to easily extend it to more complex constraints, in particular the presence of currents.

We also showed that CP techniques on continuous domains have interesting properties: (1) the use of continuous domains allows not to discretize the time, and thus to guarantee a collision-free solution; (2) in the particular case of velocity tuning, 2B-consistency techniques are computationally efficient and thus potentially usable in on-boards planners.

Future work could concern more complete experiments and the evaluation of the capabilities of higher consistencies. Higher consistencies would probably provide a sharper approximation but we have to check whether the computation cost is not too high with respect to the constraints of on-board planners.

References

1. Alias website, <http://www-sop.inria.fr/coprin/logiciels>
2. Araya, I., Trombettoni, G., Neveu, B.: Filtering numerical cSPs using well-constrained subsystems. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 158–172. Springer, Heidelberg (2009)
3. Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. In: Handbook of Constraint Proof Constraint Programming, ch. 16. Elsevier, Amsterdam (2006)
4. Botella, B., and Taillibert, P. Interlog: constraint logic programming on numeric intervals. In: International Workshop on Software Engineering, Artificial Intelligence and Expert Systems (1993)

5. Carlsson, M., Ottosson, G., Carlson, B.: An open-ended finite domain constraint solver. In: Proceedings of Programming Languages: Implementations, Logics, and Programs (1997)
6. Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*, 269–271 (1959)
7. Hentenryck, P. V., McAllester, D., Kapur, D.: Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis* 34 (1997)
8. Ibex website, <http://www.ibex-lib.org>
9. Icos website, <http://sites.google.com/site/ylebbeh/icos>
10. Ju, M.-Y., Liu, J.-H., Hwang, K.-S.: Real-time velocity alteration strategy for collision-free trajectory planning of two articulated robots. *Journal of Intelligent and Robotic Systems* 33, 167–186 (2002)
11. Lebbah, Y., Michel, C., Rueher, M., Daney, D., Merlet, J.P.: Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis* 42, 2076–2097 (2005)
12. Lhomme, O.: Consistency techniques for numeric csp. In: Proceedings of International Joint Conference on Artificial Intelligence, pp. 232–238 (1993)
13. Meteo france website, <http://marine.meteofrance.com/marine/accueil?51759.path=marine%252Fimgmervent>
14. Nelson, D., Barber, D.B., McLain, T.W., Beard, R.W.: Vector field path following for small unmanned air vehicles. In: Proceedings of the American Control Conference (2006)
15. Real paver website, <http://realpaver.sourceforge.net>
16. Soullignac, M.: Planification de trajectoire en présence de courants. Applications aux missions de drones. PhD Thesis - THALES/Nice Sophia-Antipolis university, pp. 179–203 (2009)
17. Soullignac, M., Taillibert, P.: Fast trajectory planning for multiple site surveillance through moving obstacles and wind. In: Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group, pp. 25–33 (2006)
18. Soullignac, M., Taillibert, P., Rueher, M.: Velocity tuning in currents using constraint logic programming. In: Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group, pp. 106–113 (2007)
19. Soullignac, M., Taillibert, P., Rueher, M.: Adapting the wavefront expansion in presence of strong currents. In: Proceedings of International Conference on Robotics and Automation, pp. 1352–1358 (2008)
20. van den Berg, J., Overmars, M.H.: Roadmap-based motion planning in dynamic environments. *Transactions on Robotics and Automation* 21, 885–897 (2005)

Contingency Plans for Air Traffic Management

Karl Sundequist Blomdahl, Pierre Flener, and Justin Pearson

Department of Information Technology
Uppsala University, Box 337, SE – 751 05 Uppsala, Sweden

Karl.Sundequist.Blomdahl.1559@student.uu.se,

{Pierre.Flener,Justin.Pearson}@it.uu.se

Abstract. We present two heuristics based on constraint technology that solve the problem of generating air traffic management contingency plans, which are used in the case of a catastrophic infrastructure failure within EUROCONTROL, the European Organisation for the Safety of Air Navigation. Of the heuristics presented, one is based on constraint-based local search and tabu search, and the other one is a constraint programming and large neighbourhood search hybrid algorithm. The heuristics show that it is feasible to automate the development of contingency plans, which is currently done by human experts; this is desirable for several reasons, for example it would allow the contingency plans to be generated with an increased frequency. The generated plans were evaluated, by EUROCONTROL, to be as good as the human-made ones.

1 Air Traffic Management and Contingency Planning

Air traffic management (ATM) at EUROCONTROL, the *European Organisation for the Safety of Air Navigation*, is about managing and ensuring a safe, efficient, and fair flow of air traffic, assuming a negligible amount of side-effects, such as adverse weather conditions. During normal operation, the *Central Flow Management Unit* (CFMU) of EUROCONTROL uses several stages, each in increasing detail, to satisfy its operational goals:

1. A strategic stage, taking place several months before the day of operation.
2. A pre-tactical stage that starts six days before the day of operation.
3. An online tactical stage during the day of operation. This stage is called the *air traffic flow and capacity management* (ATFCM) stage [2], and has two main functions:
 - (a) Calculate the demand of each airspace volume using live flight plan information.
 - (b) Adjust the number of allocated departure slots of the involved aerodromes, such that they optimise the objectives defined in the pre-tactical stage. These objectives typically include, but are not limited to, minimising the total flight delay and air volume overload.

During an average day, the ATFCM unit handles approximately 30 000 flights spread over about 1 500 aerodromes.

Flow identifier	Flow description	Time span	Hourly rate
EBBR1	From EBBR To C EG EI K M	00:00 – 06:00	2
		06:00 – 09:00	3
		09:00 – 12:00	7
		12:00 – 14:00	4
		14:00 – 22:00	8
		22:00 – 24:00	2
EBBR2	From EBBR To B EDDH EDDW EE EF EH EK EN ES	00:00 – 06:00	1
		06:00 – 17:00	4
		17:00 – 21:00	6
		21:00 – 24:00	2

Fig. 1. A contingency plan excerpt, which describes the hourly take-off rates of two flows originating from the aerodrome EBBR (Brussels national airport)

This study will focus on the special case of an ATFCM failure due to any reason, such as downtime of the computer-assisted slot allocation (CASA) system. In such a situation, where no timely updates from ATFCM are available and the air controllers of each aerodrome have no idea whether it is proper to release a flight or not, a safe alternative is necessary. EUROCONTROL addresses this by a *contingency plan*, which contains a pre-defined number of allocated departure slots for each major aerodrome in such a way that certain safety and efficiency objectives are satisfied, for a maximum duration of one day. During the last twelve years, such a situation has occurred once, for a few hours.

An excerpt from such a contingency plan can be seen in Figure 1. It defines the number of departure slots that the aerodrome with the *International Civil Aviation Organization* (ICAO) identifier EBBR (Brussels national airport, Belgium) is allowed to release for each hour to each destination aerodrome. For example, from 09:00 to 12:00, a maximum of 7 flights in the flow EBBR1, which is defined by the departure aerodrome EBBR and a destination aerodrome whose ICAO identifier starts with C (Canada), EG (Great Britain), EI (Ireland), K (United States), or M (Central America and Mexico) are allowed to take off. Similarly, only 4 flights whose departure and destination aerodrome match the description of the flow EBBR2 are allowed to take off per hour from 06:00 to 17:00. The current contingency plan can always be downloaded from the CFMU homepage, in the upper-left corner of <https://www.cfm.europa.eu/PUBPORTAL/gateway/spec/>.

The generation of ATM contingency plans within the *EUROCONTROL Experimental Centre* (EEC) and the CFMU is currently done by two human experts (using a process described in Section 2.2 below), who biannually develop a three-fold plan, namely one for weekdays, one for Saturdays, and one for Sundays, with a total development time of two person-months per year. Therefore, automated contingency planning is desirable. This paper presents two heuristics that solve the subproblem of finding the optimal hourly numbers of departure slots for pre-defined flows and time spans (which typically do not change much between plans anyway), and is intended as a feasibility study about whether it is possible to replace the human experts with constraint technology. Other

benefits with automating the process are that it could be done at the tactical level instead of the strategic level, which would increase the quality of the generated contingency plans.

The rest of this paper is split into five parts, each of which deals with the problem in increasingly concrete terms. In order of appearance: a formal definition of the problem (Section 2), a constraint model that implements the formal definition (Section 3), heuristics that operate on the constraint model (Section 4), experimental results (Section 5), and a conclusion (Section 6).

2 The Contingency Planning Problem

We give a detailed description of the *contingency planning problem* (CPP), the current state of the art algorithm, and a comparison with other problems.

2.1 Formal Definition

Each instance of the CPP is defined by the following input and output data, where identifiers starting with capital letters denote sets, subscripted identifiers denote constants, identifiers with indices within square brackets denote decision variables, identifiers that are Greek letters denote parameters, and all time moments are measured in seconds since some fixed origin:

- A set of flights $F = \{f_1, \dots, f_m\}$, where each flight f_ℓ has a departure aerodrome $adep_\ell$, a destination aerodrome $ades_\ell$, an expected take-off time $etot_\ell$, an expected landing time $eldt_\ell$, and a take-off delay $delay[\ell]$. All later specified sets of flights are subsets of F .
- A set of flows $\mathcal{F} = \{\mathcal{F}_1, \dots, \mathcal{F}_n\}$, where each flow \mathcal{F}_f consists of a set of flights F_f and a set of span-rate pairs $\mathcal{R}_f = \{r_1, \dots, r_{o_f}\}$, where each span-rate pair r_i consists of a time span $span_i$ for when it is active, and an hourly number of allocated departure slots $rate[i]$. Further, for any two span-rate pairs r_i and r_j , where $i \neq j$, their spans must not overlap; however, the union of all spans does not need to be 00:00–24:00. There is also a set $F_f \subseteq F$ for each \mathcal{F}_f that contains all flights matching the flow description. For example, Figure 1 defines two flows EBBR1 and EBBR2, where the flights are defined by a subset of F that matches the flow description, and the spans and rates are defined by the two right-most columns.
- A set of air volumes $AV = \{av_1, \dots, av_p\}$, where each air volume $av_a \in AV$ has a capacity cap_a that limits the hourly number of flights that can enter it for the duration dur_a . There is also a set $F_a \subseteq F$ for each av_a that contains all flights that pass through the air volume, where each flight $f_\ell \in F_a$ has an expected entering time $enter_{a,\ell}$. In the real world, an air volume can represent either a part of the airspace, or an aerodrome.

Recall that ATM has three operational goals: minimise the cost of the total flight delay, ensure a safe flow of air traffic, and ensure a fair flow of air traffic. During a crisis situation, safety is especially important.

Cost of the Total Flight Delay. The take-off delay $delay[\ell]$ of flight f_ℓ is the difference between its calculated take-off time $ctot[\ell]$ and expected take-off time $etot_\ell$, where $ctot[\ell]$ is calculated using the allocated departure slots as defined by the rate-span pairs for each flow. These slots are assigned to flights using the first-scheduled, first-served principle [3]. For example, consider the flow EBBR1 (defined in Figure 1), where there are two departure slots allocated for each hour between 00:00 and 06:00; if three flights with an $etot_\ell$ of 03:00, 03:10, and 03:20 were available, then they would get a $ctot[\ell]$ of 03:00, 03:30, and 04:00, and a delay of 0, 1200, and 2400 seconds, respectively. Similarly, each flight f_ℓ has a calculated entering time $center[a, \ell]$ into air volume av_a , which is the sum of $enter_{a, \ell}$ and $delay[\ell]$. The cost of each flight delay is defined as a weight function, which was suggested to us by our partners at the EEC:

$$delayCost[\ell] = \begin{cases} 1 & \text{if } delay[\ell] < 3600 \text{ seconds} \\ 10 & \text{if } 3600 \leq delay[\ell] < 7200 \text{ seconds} \\ 20 & \text{if } 7200 \leq delay[\ell] < 10800 \text{ seconds} \\ 50 & \text{otherwise} \end{cases}$$

The weight function scales exponentially because the real-world consequences do; for example, a flight with a low delay will probably only cause a slight interruption in the schedule, while a high delay might cause many flights to be cancelled. The cost of the total flight delay is the sum of all flight delay costs.

Air Traffic Safety. The *safety* of air traffic is determined by how crowded the air volumes are; for example the air volume av_a is capable of handling up to cap_a flights entering per hour, so any flight above this capacity creates an additional risk. Hence, safety is here defined by the amount that each air volume’s hourly capacity is exceeded. For each air volume av_a , a set T_a is defined that contains the beginning times of all one-hour-long time intervals that fit inside the air volume’s capacity duration dur_a (with a five minute step):

$$T_a = \{t \in dur_a \mid t + 3600 \in dur_a \wedge t \bmod 300 = 0\}$$

A *demand overload* is calculated for each time $t \in T_a$ as the number of flights, beyond the air volume capacity, entering the air volume during the right-open interval $[t, t + 3600)$:

$$overload[a, t] = \max(0, |\{f_\ell \in F_a \mid center[a, \ell] \in [t, t + 3600)\}| - cap_a)$$

The overload cost $overloadCost[a, t]$ of each air volume av_a and time $t \in T_a$ is a piecewise linear function of the overload percentage $\frac{overload[a, t]}{cap_a}$, where a weight is defined for the overload percentages 0%, 10%, 20%, 30%, and 40%. An illustration of this function can be seen in Figure 2. Again, the cost scales exponentially, because a small overload will likely only increase the workload of the affected ATM personnel slightly, while a large overload might result in a mistake by the ATM personnel. The cost of the total traffic demand overload is the sum of the cost for each air volume and time.

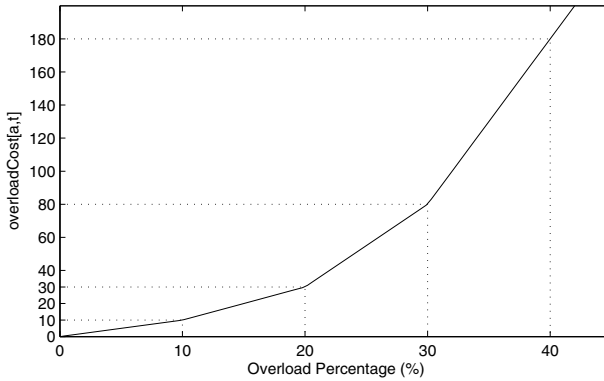


Fig. 2. An illustration of the overload cost $overloadCost[a, t]$, for the overload percentages between 0% and 40%

Air Traffic Fairness. The *fairness* of air traffic is here defined by how fairly the departure slots are allocated among the flows. No formal definition of fairness will be given at this point, as it is instead handled differently in each of our heuristics.

The Objective Function. The objective function is a linear combination of the total delay cost and the total overload cost, where α and β are parameters:

$$cost = \alpha \cdot \sum_{\ell \in F} delayCost[\ell] + \beta \cdot \sum_{a \in AV} \sum_{t \in T_a} overloadCost[a, t] \quad (1)$$

Experimental results and feedback from our partners at the EEC suggest that $\alpha = 6$ and $\beta = 1$ are good for our benchmark instances; however, they can be changed to reflect a desired balance between a low delay and a low traffic demand overload, for each problem instance.

2.2 Current State of the Art

The current state of the art, and the only known procedure, to solve the CPP is the unpublished process used by the CFMU and EEC human experts. It has been described to us in the following high-level steps:

1. A statistical analysis is performed in order to point out the airspace volumes with a high demand. The duration and capacity of each air volume are recorded (there may be several periods per volume).
2. An analysis of departing flows is made:
 - For the major European airports (i.e., with more than 2 arrivals or departures per hour on average), the traffic needs to be divided into main flows, where several destinations are grouped into each flow.

- For the other airports, flows are mainly divided into two categories: domestic flights and international flights. If the number of domestic flights is low, it seems better that a local flow manager handles this traffic.

Recall that it takes one person-month for two senior human experts to perform this procedure, and that all this is done twice a year, for weekdays, Saturdays, and Sundays.

2.3 Comparison with Other Problems

The CPP resembles several well-studied problems, especially scheduling problems. However, none of the studied problems can be used directly to solve the CPP. A case study of a few selected problems follows, in order to highlight the unique aspects of the CPP.

Cumulative Job Shop Scheduling. The *cumulative job shop scheduling problem* (CJSSP) is a well-studied multi-machine and multi-stage scheduling problem, and is proven \mathcal{NP} -hard. An instance is given by a set of activities, a set of resources AV , and a set of jobs F , where each job consists of a set of activities. Each activity $act_{a,t}$ has a processing time and a demand for each resource to be executed. Each resource $av_a \in AV$ has a capacity cap_a . A schedule is an assignment to each activity's starting time, and a schedule is feasible if it satisfies the capacity constraints for each resource, which requires that no more than cap_a units of each resource are required at once, as defined by the demand of each running activity.

Comparison with the CPP. With some extensions, such as soft capacities [6], the CJSSP is very closely related to the CPP. However, it cannot directly solve the CPP because of the relationship between the flow rates and the flight take-off delays, which causes the domain of any activity's starting time to be a function of all other activities' starting times (since two activities in the same flow must correspond to the same rate variable assignment). This complication means none of the established heuristics for solving the CJSSP can be directly applied to the CPP.

Multi-Commodity Flow. The *multi-commodity flow problem* (MCF) is a network flow problem, which given a graph $G = (V, E)$, a set of sources $S \subset V$, a set of sinks $T \subset V$, and a set of commodities C , minimises some cost as the commodities C flow over the network from source vertices to sink vertices. Since the MCF is a network flow problem, many well-studied extensions such as limited commodity quantities, restricted commodity flow paths, edge capacities, and dynamic flows (for modelling a notion of time) can be applied.

Comparison with the CPP. All state of the art solutions to the MCF are based on linear programming. This poses a problem since the objectives of the CPP are non-linear, and while rewriting them to linear expressions might be possible, no further investigation of the MCF as a modelling tool has been done yet.

3 The Constraint Model

Our constraint model implements the formal definition of the CPP using constraint technology. The model consists of five main parts: the decision variables and their domains, the problem constraints, the channelling constraints between the contingency plan and the flight delays, the channelling constraints between the flight delays and the air volume overloads, and the objective function.

3.1 The Decision Variables and Their Domains

Recall that the decision variables of the model are the identifiers that use square brackets rather than subscripts. In most cases, the domains of the decision variables can be derived from their definition; however, the following decision variables have explicitly defined domains:

- $\forall f_\ell \in F : delay[\ell] \in \mathbb{N}$ (a smaller domain is calculated from Section 3.3)
- $\forall \mathcal{F}_f \in \mathcal{F}, r_i \in \mathcal{R}_f : rate[i] \in [1, demand_{f,i}]$, where $demand_{f,i}$ is the maximum number of flights that are planned to depart in flow \mathcal{F}_f during the time span $span_i$, and T_i is defined like T_a , but for $span_i$ instead of dur_a :

$$demand_{f,i} = \max_{t \in T_i} |\{f_\ell \in F_f \mid ctot[\ell] \in [t, t + 3600)\}|$$

3.2 Problem Constraints

The following problem constraints for any flight f_ℓ and air volume av_a :

$$\begin{aligned}
 ctot[\ell] &= etot_\ell + delay[\ell] \\
 cnter[a, \ell] &= enter_{a,\ell} + delay[\ell]
 \end{aligned}$$

establish the relationships between computed and expected times.

3.3 Channelling between Contingency Plan and Flight Delays

The channelling between the contingency plan and the flight delays is defined by the mappings $\mathcal{T}_f \in A_f \rightarrow D_f$ for each flow \mathcal{F}_f , where $A_f \subseteq \mathbb{N}_0^{|\mathcal{R}_f|}$ is the Cartesian product of the flow rate decision variable domains, and $D_f \subseteq \mathbb{N}_0^{|\mathcal{F}_f|}$ are the take-off delays according to each element in A_f . For example, consider the flow \mathcal{F}_f , where $\mathcal{R}_f = \{r_1, r_2\}$, $\text{dom}(rate[1]) = \{1, 2\}$, and $\text{dom}(rate[2]) = \{3\}$; therefore $A_f = \{\langle 1, 3 \rangle, \langle 2, 3 \rangle\}$ and $\mathcal{T}_f = \{\langle 1, 3 \rangle \mapsto \langle \dots \rangle, \langle 2, 3 \rangle \mapsto \langle \dots \rangle\}$, where the actual take-off delays as calculated by $rate[1], rate[2] := 1, 3$ or $rate[1], rate[2] := 2, 3$ have been omitted for space reasons.

In classical CP (by complete tree search interleaved with propagation at every node), each mapping \mathcal{T}_f can be implemented by a *table* constraint, such that each row corresponds to one map $x \mapsto y$ in some \mathcal{T}_f . In *constraint-based local search* (CBLS), \mathcal{T}_f can instead be used as a look-up table for the take-off delays whenever a rate decision variable changes. Further details, which have been omitted for space reasons, can be found in [7].

3.4 Channelling between Flight Delays and Air Volume Overloads

The channelling between the flight delays and the air volume overloads is modelled as a *cumulative job shop scheduling problem* (CJSSP) with a time step of five minutes, where each air volume av_a is a resource, each flight f_ℓ a job, and the activities are defined by the air volumes each flight passes through. Each such activity $act_{a,\ell}$ has the following parameters:

- $resource[act_{a,\ell}] = a$
- $start[act_{a,\ell}] = enter[a, \ell] - enter[a, \ell] \bmod 300$ (this is the calculated entering time rounded down to the closest five minute tick)
- $duration[act_{a,\ell}] = 3600$ seconds (since capacity is defined hourly)
- $end[act_{a,\ell}] = start[act_{a,\ell}] + duration[act_{a,\ell}]$
- $demand[act_{a,\ell}] = 1$ unit

The capacity of each resource av_a is cap_a . Further, as the time set T_a of an air volume av_a might not cover the entire day, one must make sure any overload that occurs during a time not in T_a does not contribute to the air volume cost. There are multiple ways of doing this: the chosen method is to add at most two activities for each day, namely one starting at the beginning of the day and ending at $\min(dur_a)$ (provided it is not empty), and the other starting at $\max(dur_a)$ and ending at the end of the day (provided it is not empty), both with a demand of $-|F_a|$. Since the worst-case scenario is that all flights are in the one-hour interval starting at the same t , adding an activity with a demand of $-|F_a|$ ensures that $overload[a, t] = 0$.

Unfortunately, practice has shown that it is impossible, and sometimes undesirable, to find a solution that satisfies such a *cumulatives* constraint, i.e., the problem is often over-constrained. The chosen method is to use a *soft cumulative* constraint (inspired by [6]), which calculates the cost of each air volume av_a and time t either by using a sweep-line algorithm [1], or by explicitly defining a decision variable for each air volume and time. Which of the two approaches is better depends on the circumstances: an explicit definition allows constant-time updates of the cost when all values are fixed (and is therefore used in the CBLS heuristic described in Section 4.1), but the sweep line provides reduced memory use (and is therefore used in the large neighbourhood search heuristic described in Section 4.2).

3.5 The Objective Function

The objective function of our model, to be minimised, is (1).

4 Local Search Heuristics

Our local search heuristics operate on the model, and based on their current state try to modify their rate decision variables in such a way that the objective function is minimised. Two such heuristics have been devised, namely: (i)

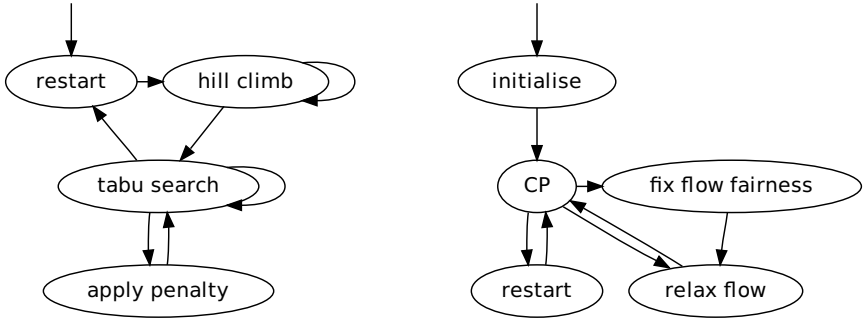


Fig. 3. To the left, the *generalised local search machine* (GLSM) [5] of our tabu search heuristic (nodes are described in Section 4.1). To the right, the GLSM of our LNS heuristic (nodes are described in Section 4.2).

a tabu search heuristic, and (ii) a *large neighbourhood search* (LNS) heuristic that uses classical *constraint programming* (CP) to search its neighbourhood. Both heuristics are described in detail in their respective sub-section, and their *generalised local search machines* (GLSM) can be seen in Figure 3. A GLSM [5] is a state machine that describes a local search heuristic by breaking it down into smaller algorithms, such that each state represents an individual algorithm and the edges represent the conditions for switching between these algorithms.

4.1 Tabu Search

Our first heuristic uses a tabu search as the core. It uses a slightly modified objective function, which adds a penalty term to (1) in order to guide the heuristic toward a fair traffic flow, where *Penalty* is a set of integer invariants:

$$cost = \alpha \cdot \sum_{\ell \in F} delayCost[\ell] + \beta \cdot \sum_{a \in AV} \sum_{t \in T_a} overloadCost[a, t] + \sum_{p \in Penalty} p$$

The heuristic can be summarised in the following steps, where each step and new terminology will be described in further details later:

1. Restart the search by assigning each $rate[i]$ a random value in its domain.
2. Hill-climb the current solution, until a local minimum has been reached.
3. Do a single iteration of tabu search, and then:
 - (a) Pick a random real number $u \in [0, 1]$; if $u < 0.05$, then pick a $rate[i]$ decision variable with an unfair value, add its penalty to *Penalty*, and go to Step 3; otherwise, do nothing and go to Step 3b.
 - (b) If more than 200 iterations have gone by since the last improvement, then go to Step 1. Otherwise, repeat Step 3.

The main source of diversity is Step 1, the main source of intensification is Step 2, and Step 3 performs mix of both.

The Restart Mechanism. The restart mechanism is the main source of diversity in the heuristic. It completely restarts the search by assigning each $rate[i]$ decision variable a random value in its domain. It also clears the tabu list.

Hill-climbing. The hill climbing algorithm is a non-greedy algorithm. During each iteration, it picks the first $\langle rate[i], v \rangle$ move such that the objective function is decreased, until no such move can be found, i.e., a local minimum has been reached. The method used to find this assignment is through the use of a meta-neighbourhood, which is a circular list of neighbourhoods $\{N_1, \dots, N_q\}$, where q is the number of rate decision variables, that are searched in successive order until an improving assignment is found, where each neighbourhood N_i corresponds to all variable-value pairs in $\{rate[i]\} \times \text{dom}(rate[i])$. The algorithm terminates once a cycle has been completed with no improving assignment found.

Tabu Search. The tabu search is the core of the heuristic. While it is the main contributor of neither intensity nor diversity, it ensures that the tabu search neighbourhood of a local minimum has been properly explored and no improvements have been missed. During each iteration, it searches a neighbourhood (to be defined later) for a best non-taboo move $\langle rate[i], v \rangle$ and, after making the inverse move taboo for the number of iterations defined by the tabu tenure, it does the assignment $rate[i] := v$. The only exception to this process is the aspiration criterion, which kicks in if the candidate solution is better than any solution found so far. If this is the case, then the move is performed even if it is in the tabu list. The current implementation uses a tabu tenure of $\tau = 8$.

The tabu search uses an asymmetrical stochastic neighbourhood that is designed to reduce the most severe overloads. It does so by finding the peak of each air volume demand overload, and then picks one of these peaks to reduce at random, where the probability of each peak being picked is proportional to its overload, hence higher peaks have a higher probability to be reduced. Once a peak has been determined, all flows \mathcal{F}_f that contain a flight contributing to this peak (flights that cannot be anywhere else can be ignored) have all $\{rate[i]\} \times \text{dom}(rate[i])$, where $r_i \in \mathcal{R}_f$, added to the current neighbourhood.

Penalty Invariant. The apply penalty state is the part of the heuristic that tries to ensure a high level of air traffic fairness. It does so by modifying the cost function at random points in time, such that the rate variable $rate[i]$ with the minimum $\frac{rate[i]}{demand_{f,i}}$ quotient is deemed unfair and an expression that tries to guide $rate[i]$ toward a fairer value is added to *Penalty*. It is an exponential expression that decreases the higher the value of $rate[i]$:

$$\gamma \cdot e^{-8 \cdot \frac{rate[i]}{demand_{f,i}}}$$

where γ is a parameter that controls how aggressively the heuristic should be guided toward fairness; the current implementation uses $\gamma = 200$, which is only slightly aggressive.

4.2 Large Neighbourhood Search

Our second heuristic is a hybrid heuristic based on classical *constraint programming* (CP) and *large neighbourhood search* (LNS). Given a feasible solution, LNS works by *relaxing* part of the solution, that is, it picks some decision variables and restores their domains to their initial values, and uses constraint programming to search the resulting search space for a better solution. Our LNS heuristic can be summarised in the following steps, where each step and new terminology will be described in further details later:

1. Set each $rate[i]$ decision variable to the maximum of its domain, and go to Step 3 in *solve* mode.
2. If in *solve* mode, use CP to find a feasible solution; else (in *optimise* mode) use CP to find a feasible solution with the minimum cost.
3. Select a $rate[i]$ decision variable from a fixed circular list that contains all rate decision variables, in an arbitrary order:
 - (a) If a full circle in the list has gone by with no improvement, then restore the domains of all rate variables, post a constraint that any next solution must be better than the current best, and go to Step 2 in *solve* mode.
 - (b) If $rate[i]$ is unfair, then post a constraint that $rate[i]$ must be fair, relax the neighbourhood of $rate[i]$ according to Step 3c, and then go to Step 2 in *optimise* mode.
 - (c) Relax the neighbourhood of $rate[i]$ using an algorithm based on maximum set coverage, post a constraint that a solution must be better than the current best, and go to Step 2 in *optimise* mode.

Constraint Propagation & Search. The CP state uses constraint propagation and search to find feasible solutions. It can do this in two available modes of operation: (i) the *solve* mode, in which it returns the first feasible solution, and (ii) the *optimise* mode, in which it exhaustively searches for the best solution using a depth-first search tree traversal. Which mode it uses depends on which was requested by the incoming call (edge in Figure 3); it does not use any internal heuristics to determine which is better. The branching heuristic used is to pick a variable $rate[i]$ at random and the value $\max(\text{dom}(rate[i]))$. The reason for this is that when searching a relaxed neighbourhood most of the search is done using propagation rather than branching, hence even if a more complicated heuristic were used not much improvement could be found.

Restart Strategy. The restart strategy, which is triggered when all neighbourhoods have been searched and no improving move has been found, restores the domains of all decision variables of all flows. It also removes any constraints added by the heuristic, except for the constraint that any next solution must be better than the current best.

Relaxation. Relaxation is the most important part of the heuristic, as it defines the neighbourhood searched during each iteration. This neighbourhood is

a cyclic list of neighbourhoods $\{N_1, \dots, N_n\}$, where each neighbourhood N_f is designed to relax the decision variables closely interconnected with flow \mathcal{F}_f . This interconnectivity is defined by the number of air volumes that two flows have in common. In more detail, for each flow \mathcal{F}_f , a set S_f is defined that contains all air volumes that some flight in F_f passes through:

$$S_f = \{av_a \in AV \mid F_a \cap F_f \neq \emptyset\} \quad (2)$$

The interconnectivity of the flows \mathcal{F}_f and \mathcal{F}_h is then defined as $|S_f \cap S_h|$, the number of common air volumes that flights in \mathcal{F}_f and \mathcal{F}_h pass through. However, more than one flow with high interconnectivity is necessary for a good neighbourhood: what is desired is to give \mathcal{F}_f a certain degree of freedom such that it can actually change in a meaningful way when relaxed; hence the neighbourhood of a flow \mathcal{F}_f is defined as the *maximum set coverage* (MSC) of the set S_f and the set collection $S \setminus \{S_f\}$, where $S = \{S_1, \dots, S_n\}$, with the slight modification that rather than limiting the number of sets that can be chosen, as is typically done in the MSC problem, the *size* of the resulting search space is instead limited, i.e., $\prod_{V \in S'} \prod_{i \in \mathcal{R}_V} |\text{dom}(\text{rate}[i])|$, where $S' \subseteq S$ contains S_f and the selected sets from S , and \mathcal{R}_V is the set \mathcal{R}_f such that S_f is V . Luckily, in practice the interconnectivity between the sets in S seems to be high, hence this is not a very hard problem to solve. Using a greedy algorithm for solving MSC problems is sufficient to produce on average over 90% coverage when using a search space limit of $\delta = 100\,000$ candidate solutions.

During each iteration, the greedy algorithm maintains two auxiliary sets: (i) $U \subseteq S_f$, which are the still uncovered elements of S_f , and (ii) $S' \subseteq \{S_1, \dots, S_n\}$, which are the sets picked as neighbours of S_f . Then, as long as S is not empty, it picks a set $V \in S \setminus S'$ with the largest intersection with U (i.e., $|V \cap U|$), where ties are broken by the largest intersection with S_f . Then, the auxiliary sets are updated, such that all elements in V are removed from U , and V is added to S' , unless doing so would make the solution space larger than the limit δ , in which case V is instead discarded. Note that the U set can be empty during an iteration; this is the reason for the lexicographic comparison when selecting a $V \in S \setminus S'$. This algorithm can be seen in Algorithm [11](#).

Returning to the relaxation state, once S' has been determined, the neighbourhood N_f is defined for all decision variables $\text{rate}[i]$, where $r_i \in \mathcal{R}_f$ and $S_f \in S'$. Then this neighbourhood is relaxed by restoring the domain of each of the variables in N_f to its initial value, followed by adding two constraints: (i) the cost of any next solution must be smaller than the current best, and (ii) for each air volume, its maximum overload must not be larger than the maximum overload of the same air volume in the current best solution. The first constraint is a standard optimisation technique, whereas the second is there to improve the propagation and to allow proper energy feasibility calculations in the *soft cumulative* constraint [6](#).

Flow fairness. The fix flow fairness state addresses any unfair values assigned to rate variables. It does this by adding a couple of constraints when a flow that

Algorithm 1. The greedy maximum set coverage algorithm used to determine the *maximum set coverage* (MSC) of flow \mathcal{F}_f , with a maximum solution space size δ

```

1: Calculate the set collection  $S = \{S_1, \dots, S_n\}$  according to (2) for each flow in
    $\mathcal{F} = \{\mathcal{F}_1, \dots, \mathcal{F}_n\}$ .
2:  $S' \leftarrow \{S_f\}$ 
3:  $S \leftarrow S \setminus \{S_f\}$ 
4:  $U \leftarrow S_f$ 
5: while  $S \neq \emptyset$  do {Invariant:  $U \subseteq S_f \wedge S \cap S' = \emptyset$ }
6:   Select the set  $V$  among  $S$  with the maximum  $\langle |U \cap V|, |S_f \cap V| \rangle$ .
7:    $S'' \leftarrow S' \cup \{V\}$ 
8:   if search space size of  $S'' < \delta$  then
9:      $S \leftarrow S \setminus \{V\}$ 
10:     $S' \leftarrow S''$ 
11:     $U \leftarrow U \setminus V$ 
12:   else
13:      $S \leftarrow S \setminus \{V\}$ 
14: return  $S'$ 

```

has a rate variable with an unfair value compared to all other flows is selected for relaxation. A value is *unfair* if it has a statistically outlying $q_i = \frac{\text{rate}[i]}{\text{demand}_{f,i}}$ quotient compared to all other flows. A value q_i is a statistical outlier if:

$$q_i \notin [E(q) - \text{std}(q), E(q) + \text{std}(q)]$$

where $E(x)$ is the expected value of the set x and $\text{std}(x)$ its standard deviation. If q_i is a statistical outlier, then a constraint requiring that $E(q) - \text{std}(q) \leq q_i \leq E(q) + \text{std}(q)$ is added; the neighbourhood of \mathcal{F}_f , where $r_i \in \mathcal{R}_f$, is relaxed as previously described, and a solution is sought in *optimise* mode. Note that no constraint requiring the solution to be better than the current best is added, because fairness is more important than a low cost.

5 Experimental Results

EUROCONTROL maintains two yearly timetables, one for the summer and one for the winter. Further, in each timetable weekdays, Saturdays, and Sundays have distinct traffic patterns. We have been provided, by the EEC, three real-life problem instances from the summer 2008 timetable that represent worst case scenarios for each distinct traffic pattern and are comparable to those used by EUROCONTROL when generating the official contingency plans:

- A Friday (June): 261 flows (320 rates), 36 161 flights, 348 air volumes.
- A Saturday (August): 256 flows (387 rates), 29 842 flights, 348 air volumes.
- A Sunday (August): 259 flows (397 rates), 31 024 flights, 348 air volumes.

Table 1. The experimental results of the different algorithms

Contingency Plan	$E(\text{delay}[f])$	$p_{95}(\text{delay}[f])$	$E(\text{overload})$	$p_{95}(\text{overload})$
EEC 2008-06-27	645.6 sec	2340.0 sec	29%	100%
EEC 2008-08-30	528.1 sec	1800.0 sec	23%	61%
EEC 2008-08-31	407.0 sec	1500.0 sec	29%	68%
tabu 2008-06-27	310.2 sec	1200.0 sec	27%	72%
tabu 2008-08-30	316.1 sec	1200.0 sec	22%	56%
tabu 2008-08-31	345.9 sec	1264.5 sec	24%	57%
LNS 2008-06-27	535.5 sec	2185.0 sec	29%	100%
LNS 2008-08-30	512.1 sec	1800.0 sec	23%	60%
LNS 2008-08-31	504.1 sec	1628.0 sec	34%	100%

When translated into a constrained optimisation problem, each instance yields approximately 150 000 constraints and 50 000 decision variables. All experimental results were done on a Linux x86-64 dual-core laptop with 4GB of primary memory, 2MB of L2 cache, and a CPU frequency of 2.2GHz. The tabu search heuristic have been implemented in Comet [8] version 2.0.1, and the LNS heuristic using Gecode [4] version 3.3.1. The tabu search usually terminated after approximately three CPU hours, while the LNS heuristic was interrupted after one CPU week (details below).

A comparison between our heuristics and a few contingency plans generated by the EUROCONTROL human experts (denoted by EEC) can be seen in Table 1, where the cost is presented as the expected take-off delay, the 95th percentile of the take-off delay, the expected air volume overload percentage (where overloads equal to zero have been omitted), and the 95th percentile of the air volume overload percentages (where overloads equal to zero have been omitted).

The first observation that can be made is that our heuristics decrease both the take-off delay and the air volume overload of the contingency plans generated by the EUROCONTROL human experts; this was expected, due to the similarities between the CPP and scheduling problems, which have been solved successfully using constraint technology for decades. However, the observation that our tabu search heuristic performs better than our LNS heuristic was unexpected, because the neighbourhood of the tabu search is a subset of the LNS neighbourhood, and should therefore perform at least as well as the tabu search heuristic. This performance difference has been attributed to the lack of runtime for the LNS heuristic, which was interrupted before reaching a local minimum, even after one week of runtime; further, this lack of runtime can probably be attributed to an inefficient implementation rather than a fault in our heuristic. However, regardless of the difference in performance between our heuristics, they show the feasibility of solving the CPP using constraint technology. The relative performance of our heuristics has been reproduced by the EEC, using their internal validation tool COSAAC and one of the current human planners. They compared our and their contingency plans on realistic test flight plans (not given to us), though *not* according to the objective function we used during the

optimisation, but more realistically according to a CASA-style slot allocation, as if CASA was actually *not* down.

6 Conclusion

This work is intended as a feasibility study about whether it is possible to automate the development of contingency plans for EUROCONTROL, the *European Organisation for the Safety of Air Navigation*. Based on the experimental results, it seems to be possible efficiently with constraint technology. Recall that this paper addresses the subproblem of finding the optimal number of allocated departure slots for predefined flows and time spans. The latter have been produced by human experts, and do not change much from one year to another. However, the dependency on predefined flows and time spans must be eliminated. Currently, this is our most important issue; ideally the search for the optimal set of flows and time spans could be integrated into our heuristics.

Acknowledgements

This work has been financed by the European Organisation for the Safety of Air Navigation (EUROCONTROL) under its Care INO III programme (grant 08-121447-C). The content of the paper does not necessarily reflect the official position of EUROCONTROL on the matter. We especially thank S. Manchon, E. Petit, and B. Kerstenne at the EUROCONTROL Experimental Centre.

References

1. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
2. EUROCONTROL. Air Traffic Flow & Capacity Management Operations ATFCM Users Manual. 14.0 edition (2010), http://www.cfm.eurocontrol.int/j_nip/cfm/public/standard_page/librar_handbook_supplements.html.
3. EUROCONTROL. General & CFMU Systems. 14.0 edition (2010), http://www.cfm.eurocontrol.int/j_nip/cfm/public/standard_page/library_handboo_supplements.html
4. Gecode Team. Gecode: Generic constraint development environment (2006), <http://www.gecode.org>
5. Hoos, H., Stützle, T.: Stochastic Local Search: Foundations & Applications. Morgan Kaufmann Publishers Inc., San Francisco (2004)
6. Petit, T., Poder, E.: Global propagation of practicability constraints. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 361–366. Springer, Heidelberg (2008)
7. Sundequist Blomdahl, K.: Contingency Plans for Air Traffic Management. Master's thesis, Uppsala University (2010), <http://www.it.uu.se/research/group/astra/>
8. Van Hentenryck, P., Michel, L.: Constraint-Based Local Search. MIT Press, Cambridge (2005)

Author Index

- Ahmadizadeh, Kiyan 514
Allouche, David 53
Araya, Ignacio 61
Asaf, Sigal 24
- Bacchus, Fahiem 176
Balafoutis, Thanasis 69
Bauer, Andreas 522
Beck, J. Christopher 84
Beldiceanu, Nicolas 137
Bent, Russell 99
Bessiere, Christian 114
Boizumault, Patrice 552
Botea, Viorica 522
Brown, Mark 522
- Cambazard, Hadrien 129
Carlsson, Mats 460
Castañeda Lozano, Roberto 613
Chabert, Gilles 137, 491
Cho, Jeremy 176
Connors, Daniel P. 24
Cooper, Martin C. 152
Crémilleux, Bruno 552
- Dal Palù, Alessandro 167
Davern, Paul 460
Davies, Jessica 176
de Givry, Simon 53
Deville, Yves 191
Dilkina, Bistra 514
Doğru, Ali 537
- Eran, Haggai 24
Ermon, Stefano 38
Eveillard, Damien 221
- Feldman, Jacob 460, 568
Flener, Pierre 643
- Gent, Ian P. 206
Goldsztejn, Alexandre 221
Gomes, Carla P. 38, 514
Gotlieb, Arnaud 330
Gray, Matt 522
- Greco, Gianluigi 236
Gresh, Donna L. 24
Guo, Long 252
- Hamadi, Youssef 252
Harabor, Daniel 522
Hoda, Samid 266
Hooker, J.N. 266
Hosobe, Hiroshi 221
- Jabbour, Said 252
Jain, Siddhartha 281
Jeavons, Peter 398
Jefferson, Chris 206
Junttila, Tommi 297
- Karataş, Ahmet Serkan 537
Kaski, Petteri 297
Katsirelos, George 114, 305
Khiari, Mehdi 552
Kolaitis, Phokion G. 475
Kotthoff, Lars 321
- Lazaar, Nadjib 330
Lebbah, Yahia 330
Lesaint, David 583
Little, James 568
Lombardi, Michele 383
- Madelaine, Florent 345
Marre, Bruno 360
Martin, Barnaby 368
McCinnis, Michael J. 24
Mehta, Deepak 460, 583
Michel, Claude 360
Michel, Laurent 598
Miguel, Ian 206, 321
Milano, Michela 383
Möhl, Mathias 167
Mullier, Olivier 221
- Narodytska, Nina 114, 305
Neveu, Bertrand 61

- Nieuwenhuis, Robert 1
 Nightingale, Peter 206, 321
 O'Mahony, Eoin 281
 Ortega, Julio 24
 O'Sullivan, Barry 129, 583
 Oğuztüzün, Halit 537
 Paparrizou, Anastasia 69
 Papegay, Yves 491
 Pearson, Justin 643
 Petke, Justyna 398
 Pourtallier, Odile 491
 Quesada, Luis 460, 583
 Quimper, Claude-Guy 114
 Richter, Yossi 24
 Rintanen, Jussi 414
 Rueher, Michel 628
 Sabharwal, Ashish 514
 Sais, Lakhdar 252
 Scarcello, Francesco 236
 Schiex, Thomas 53
 Schreiber, Yevgeny 429
 Schulte, Christian 613
 Schutt, Andreas 445
 Sellmann, Meinolf 281
 Selman, Bart 38
 Shvartsman, Alexander A. 598
 Simonis, Helmut 460
 Slaney, John 522
 Sonderegger, Elaine 598
 Soullignac, Michaël 628
 Stergiou, Kostas 69
 Stynes, David 568
 Sundequist Blomdahl, Karl 643
 Taillibert, Patrick 628
 Tan, Wang-Chiew 475
 ten Cate, Balder 475
 Trombettoni, Gilles 61, 491
 Tsang, Edward 3
 van der Krogt, Roman 568
 Van Hentenryck, Pascal 99, 191,
 499, 598
 van Hove, Willem-Jan 266
 Vardi, Moshe Y. 8
 Wahlberg, Lars 613
 Walsh, Toby 69, 114, 305
 Willard, Ross 9
 Will, Sebastian 167
 Wilson, Nic 583
 Wolf, Armin 445
 Yip, Justin 499
 Živný, Stanislav 152