Department of Computer Science
Series of Publications A
Report A-2018-5

Solving Optimization Problems via Maximum Satisfiability: Encodings and Re-Encodings

Jeremias Berg

To be presented with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium CK112, Exactum, Gustaf Hällströmin katu 2b, on May 25th, 2018, at 12 o'clock noon.

University of Helsinki
Finland

Supervisors

Associate Professor Matti Järvisalo, University of Helsinki, Finland
Professor Petri Myllymäki, University of Helsinki, Finland

Pre-examiners

Professor Lakhdar Sais, Université d'Artois, France
Professor Peter Stuckey, University of Melbourne, Australia

Opponent

Associate Professor Inês Lynce, Universidade de Lisboa, Portugal

Custos

Associate Professor Matti Järvisalo, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: https://www.helsinki.fi/en/computer-science
Telephone: +358 2941 911

Copyright © 2018 Jeremias Berg
ISSN 1238-8645
ISBN 978-951-51-4241-2 (paperback)
ISBN 978-951-51-4242-9 (PDF)
Helsinki 2018
Unigrafia

Solving Optimization Problems via Maximum Satisfiability: Encodings and Re-Encodings

Jeremias Berg

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
jeremiasberg@gmail.com
http://www.jeremiasberg.com

PhD Thesis, Series of Publications A, Report A-2018-5
Helsinki, April 2018, 86 + 102 pages
ISSN 1238-8645
ISBN 978-951-51-4241-2 (paperback)
ISBN 978-951-51-4242-9 (PDF)

Abstract

NP-hard combinatorial optimization problems are commonly encountered in numerous different domains. As such efficient methods for solving instances of such problems can save time, money, and other resources in several different applications. This thesis investigates exact declarative approaches to combinatorial optimization within the maximum satisfiability (MaxSAT) paradigm, using propositional logic as the constraint language of choice. Specifically we contribute to both MaxSAT solving and encoding techniques.

In the first part of the thesis we contribute to MaxSAT solving technology by developing solver independent MaxSAT preprocessing techniques that re-encode MaxSAT instances into other instances. In order for preprocessing to be effective, the total time spent re-encoding the original instance and solving the new instance should be lower than the time required to directly solve the original instance. We show how the recently proposed label-based framework for MaxSAT preprocessing can be efficiently integrated with state-of-art MaxSAT solvers in a way that improves the empirical performance of those solvers. We also investigate the theoretical effect that label-based preprocessing has on the number of iterations needed by MaxSAT solvers in order to solve instances. We show that preprocessing does not improve best-case performance (in the number of iterations) of MaxSAT solvers, but can improve the worst-case performance. Going beyond previously proposed preprocessing rules we also propose and evaluate a MaxSAT-specific preprocessing technique called subsumed label elimination (SLE). We show that SLE is theoretically different from previously proposed MaxSAT preprocessing rules and that using SLE in conjunction with other preprocessing rules improves empirical performance of several MaxSAT solvers.

In the second part of the thesis we propose and evaluate new MaxSAT encodings to two important data analysis tasks: correlation clustering and bounded treewidth Bayesian network learning. For both problems we empirically evaluate the resulting MaxSAT-based solution approach with other exact algorithms for the problems. We show that, on many benchmarks, the MaxSAT-based approach is faster and more memory efficient than other exact approaches. For correlation clustering, we also show that the quality of solutions obtained using MaxSAT is often significantly higher than the quality of solutions obtained by approximative (inexact) algorithms. We end the thesis with a discussion highlighting possible further research directions.

Computing Reviews (2012) Categories and Subject Descriptors:

Mathematics of computing→Combinatorial optimization
Theory of computation→Constraint and logic programming
Theory of computation→Problems, reductions and completeness

General Terms:
Algorithms, Satisfiability, Combinatorial Optimization

Additional Key Words and Phrases:
constraint optimization, maximum satisfiability, MaxSAT, preprocessing

Acknowledgements

This work was done as part of the Constraint Reasoning and Optimization (CoReO) group of the Department of Computer Science at the University of Helsinki. First and foremost I would like to express my sincerest gratitude to my advisor, Associate Professor Matti Järvisalo for all of the guidance and advice I have received during my PhD studies. I especially want to thank you for believing in me enough to let me pursue my own research interests. I am also very grateful to Professor Petri Myllymäki for the numerous roles he has played in making this work possible. Many thanks go to all of my other co-authors as well: Paul Saikko, Brandon Malone, Tuukka Korhonen, Antti Hyttinen, Emilia Oikarinen, Kai Puolamäki, Kerstin Bunte and Samuel Kaski. Working with all of you has been most pleasant and educational.

The quality of this manuscript has been significantly improved by the valuable feedback I have received from various people. I would especially like to thank my pre-examiners Professor Lakhdar Sais and Professor Peter Stuckey for their valuable input as well as Associate Professor Inês Lynce for taking the time to come to Helsinki to be my opponent. I extend my gratitude to everyone else who has given me feedback and suggestions, Matti, Antti, Jonas and Brandon as well as all anonymous reviewers of each publication.

I am immensely grateful for the support I have received from the Doctoral School of Computer Science (DoCS). I thank the board and steering committee for giving me the financial stability needed complete my PhD. I especially want acknowledge Dr Pirjo Moen for helping me with most non-research related issues during my time as a PhD student. I also want to thank the Emil Aaltonen Foundation and the Nokia Foundation for financially supporting my PhD research.

I consider myself very lucky to have been a member of the CoReO research group. I'd like to thank past and present members of CoReO for the very inspiring research environment you have provided and the many interesting discussions we have had over the years. In addition to my colleagues I would also like to thank my friends and other colleagues, both in and outside of Kumpula campus, for the welcome and necessary distractions from research. A special thanks goes to everyone that played cards with me in the coffee room or belayed me on the climbing wall. I am also immensely grateful for the opportunities given to me by my family and relatives. Without your support I would not be who I am today and probably would not have pursued a PhD in the first place. Finally I would like to thank Christina for being there for me through the ups and downs of day to day life.

Helsinki, April 2018

Jeremias Berg

Contents

Chapter 1

Introduction

Mathematical optimization is a rich field of study with numerous applications. Whenever we are given a problem and tasked with finding a solution that is “best”, we are faced with an optimization problem. If the space of possible (feasible) solutions is discrete, we talk about a combinatorial optimization problem [1]. The exact definition of a solution being best (optimal) depends on the specific problem at hand. Commonly used quality measures include the length or cost of a solution. In this thesis, we focus on computationally challenging combinatorial optimization problems and, in particular, on developing maximum satisfiability [2] as a tool for solving them.

Computationally challenging optimization problems are common. Several of the well-known NP-complete decision problems correspond to NP-hard optimization problems. Consider, for example, the *traveling salesperson problem* (TSP) [3, 4]. An instance of TSP consists of a set of locations and the pairwise distances between them. A (feasible) solution to the instance is a route which visits all of the locations. The problem of deciding the existence of a route that has length at most some given bound is NP-complete. The corresponding NP-hard combinatorial optimization problem asks to find the shortest possible route.

NP-hard optimization problems are encountered in various settings, including, but not restricted to: telecommunications and network design [5], computational biology [6, 7], clustering [8–10], structure learning of probabilistic graphical models [11–13], argumentation [14], itemset mining [15–18], data visualization [19–21], planning [22–24], scheduling [25–30], routing [31], timetabling [32–36], hardware and software verification [37–39], covering [40], air traffic management [41, 42] and cancer therapy design [43].

The abundance and diversity of optimization problems suggests that efficient algorithms for can save time (e.g,. scheduling), money (e.g., network design) or other resources in various applications. For example, an effective solution method to TSP could significantly decrease the delivery times and fuel costs of a delivery company.

The research field of combinatorial optimization is well-established and studied [1]. The solution approaches to combinatorial optimization problems can roughly be divided into four categories: approximation algorithms [44–48], local search algorithms [49–52], problem-specific exact algorithms [3, 53–56] and exact declarative methods [2, 57–62]. This thesis focuses on exact declarative methods for solving NP-hard combinatorial optimization problems.

Figure 1.1 overviews the declarative approach to solving an instance p of an NP-hard optimization problem $\mathcal{P}$. The first step of the declarative approach is the *encoding* of p into some mathematical constraint language $\mathcal{L}$. In other words, the declarative approach assumes the existence of a function (an encoding) $\mathcal{P} \to \mathcal{L}$ that maps each instance p of $\mathcal{P}$ to an instance $\mathcal{F}(p)$ of $\mathcal{L}$, i.e., a set of constraints in $\mathcal{L}$. The instance $\mathcal{F}(p)$ describes p in the sense that optimal solutions to $\mathcal{F}(p)$ correspond to optimal solutions to p. We assume that the constraint instance $\mathcal{F}(p)$ can be formed in polynomial time with respect to the size of p. This assumption is typical when working with declarative methods, although there has been some research into larger encodings, often for solving even more complex problems [63–65].

After encoding p into $\mathcal{F}(p)$, the next step in the declarative approach is *solving* $\mathcal{F}(p)$, i.e., computing an optimal assignment to the variables in $\mathcal{F}(p)$. We call such an assignment an *optimal solution* to $\mathcal{F}(p)$. Finally, the optimal solution to $\mathcal{F}(p)$ is used to reconstruct an optimal solution to p. Analogously to the encoding step, we assume that the reconstruction step is computable in polynomial time. Since $\mathcal{P}$ is NP-hard, these assumptions imply that $\mathcal{L}$ should be NP-hard as well. More specifically, we focus in this thesis on optimization problems and constraint languages with NP-complete decision counterparts. In the rest of the text, we use the term NP-hard in an informal manner to refer to specifically to such problems.

A notable characteristic of the pipeline in Figure 1.1 is that the (only) two computationally challenging steps are defining an encoding $\mathcal{P} \to \mathcal{L}$ and solving the constraint instance $\mathcal{F}(p)$. Assuming P $\neq$ NP, no complete solver for an NP-hard constraint language will run in polynomial time on every instance [66]. The efficiency of the declarative approach relies instead on designing solvers and encodings which ensure that the "interesting" instances of $\mathcal{P}$ are encoded into constraint instances $\mathcal{F}(p)$ on which the solver is able to avoid its worst case running time. By interesting instances we mean instances that are encountered in actual applications of the prob-

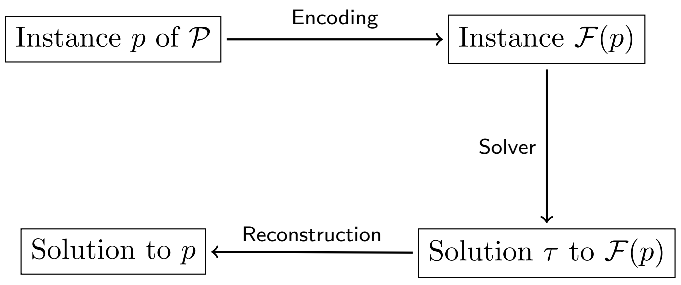
Instance p of $\mathcal{P}$
Encoding
Instance $\mathcal{F}(p)$
Solver
Reconstruction
Solution to p
Solution τ to $\mathcal{F}(p)$

Figure 1.1: A declarative approach to solving an optimization problem $\mathcal{P}$.

lem. Consider for example a delivery company applying a solution method to TSP. A significant fraction of the theoretically possible instances (sets of locations) of TSP are never going to be encountered by the company in practice. Instead, an encoding and a solver which together are able to solve the instances corresponding to actually possible locations are enough to obtain a solution approach to TSP which is sufficient for the company's needs.

A significant benefit of the declarative approach to solving optimization problems is its generality. The computationally challenging step of solving $\mathcal{F}(p)$ is independent of the particular optimization problem $\mathcal{P}$ being solved. This means that improvements in solver technology of the chosen constraint language translate directly into more efficient algorithms to several different optimization problems, given the existence of well-performing encodings. Over the last decades, a number of different NP-hard constraint languages with varying features have been proposed and developed. A well known example is integer programming [57, 67, 68]. Others include constraint programming [59, 69], answer set programming [60, 61], maximum satisfiability [2] and its extensions to satisfiability modulo theories [70–72]. This thesis focuses on propositional logic as the underlying constraint language and maximum satisfiability as the corresponding constraint optimization problem.

1.1 Maximum Satisfiability

Maximum satisfiability (MaxSAT) is the optimization counterpart of the archetypical NP-complete propositional satisfiability (SAT) problem [66]. The expressive semantics of propositional logic, the constraint language underlying MaxSAT, allow encoding many NP-hard optimization problems as MaxSAT instances. At the same time, the relatively simple syntax also allows the development of efficient solvers. The potential of propositional logic as the constraint language has been witnessed by the exceptional success of SAT solvers over the last decade [73, 74]. Recent improvements in MaxSAT solving technology and encodings have led to MaxSAT being applied in many different problem domains, including clustering [75], probabilistic modeling [76–79], data visualization [20], haplotype inference [80–82], game theory [83] treewidth computation [84], reasoning over biological networks [85, 86], electronic markets [87], routing [31], software verification and code debugging [37, 88–92], planning [24, 93, 94], cancer therapy design [43], computing covering arrays [95] scheduling [36], probabilistic reasoning [78], upgradeability [96], design debugging [97], analysis of other constraint satisfaction problems [98] and computer memory reconstruction [99].

The state of the art in MaxSAT solving techniques is evaluated annually in the MaxSAT Evaluations [100–102]. The evaluations have shown that the effectiveness of MaxSAT solvers for solving other optimization problems builds heavily on the effectiveness of SAT solvers. More specifically, many of the solvers that are most effective on MaxSAT instances that correspond to other optimization problems make extensive use of satisfiability solvers as subroutines. In the rest of the thesis such solvers are collectively called SAT-based MaxSAT solvers. SAT-based MaxSAT solvers can further be divided into roughly three subcategories: the model-guided [103–108], the core-guided [106, 109–120] and the implicit hitting set based [121–123] solvers. Most of the contributions of this thesis are developed in the context of core-guided and implicit hitting set based solvers, although many of the ideas are simple to extend to model-guided solvers as well.

It should be noted that in addition to SAT-based MaxSAT solvers, another commonly used approach to MaxSAT solving is branch and bound (B&B) [124–134]. B&B solvers tend to be most effective on random MaxSAT instances as well as challenging instances of smaller size. Such instances are encountered for example in combinatorics [100–102].

1.2 Contributions of the Thesis

This thesis is based on six peer-reviewed publications. The contributions of this thesis are divided into two interrelated research questions. In this section we first overview the publications and then discuss the research questions. We also briefly overview the specific contributions of the present author to each individual publication. The remaining chapters of the thesis will then discuss the contributions of each publication in more detail.

1.2.1 Original Publications

The following six peer-reviewed publications form the basis of this thesis. The papers are referred to as Papers I-VI in the rest of the text.

I Jeremias Berg, Paul Saikko, and Matti Järvisalo. **Improving the Effectiveness of SAT-Based Preprocessing for MaxSAT**. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 239-245. AAAI Press, 2015.

II Jeremias Berg, Paul Saikko, and Matti Järvisalo. **Re-using Auxiliary Variables for MaxSAT Preprocessing**. In *Proceedings of the IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 813-820. IEEE Computer Society, 2015.

III Jeremias Berg and Matti Järvisalo. **Impact of SAT-Based Preprocessing on Core-Guided MaxSAT Solving**. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming (CP)*, volume 9892 of Lecture Notes in Computer Science, pages 66-85. Springer International Publishing, 2016.

IV Jeremias Berg, Paul Saikko, and Matti Järvisalo. **Subsumed Label Elimination for Maximum Satisfiability**. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, volume 285 of Frontiers in Artificial Intelligence and Applications, pages 630-638. IOS Press, 2016.

V Jeremias Berg and Matti Järvisalo. **Cost-Optimal Constrained Correlation Clustering via Weighted Partial Maximum Satisfiability**. *Artificial Intelligence.* 244:110-142, 2017.

VI Jeremias Berg, Matti Järvisalo, and Brandon Malone. **Learning Optimal Bounded Treewidth Bayesian Networks via Maximum Satisfiability**. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 33 of JMLR Workshop and Conference Proceedings, pages 86-95. JMLR, 2014.

Reprints of the publications are included at the end of the thesis.

1.2.2 Research Questions

This thesis contributes to improving the effectiveness of using MaxSAT for solving combinatorial optimization problems by studying two distinct but connected research questions. The first question concerns the development of MaxSAT solving methods, specifically in the form of solver-independent MaxSAT preprocessing [135]. The second question concerns the development of new MaxSAT encodings for two data analysis problems, correlation clustering [136] and bounded treewidth Bayesian network structure learning [137, 138].

Research Question 1: Preprocessing in MaxSAT solving

The first part of the thesis focuses on improving SAT-based MaxSAT solving. More specifically, Papers I-IV develop *preprocessing* techniques for MaxSAT. Preprocessing [139–141] extends the declarative pipeline (Figure 1.1) by adding a preprocessing step directly after the encoding step. During preprocessing, the constraint instance $\mathcal{F}(p)$ is re-encoded into another constraint instance $pre(\mathcal{F}(p))$ using polynomial-time computable inference rules. In this context, the inference rules are called *preprocessing rules* and the process of re-encoding $\mathcal{F}(p)$ is called *preprocessing* $\mathcal{F}(p)$. Analogously to the encoding, the preprocessing rules used should preserve optimal solutions. Informally, we say that preprocessing is *sound* if any optimal solution to $pre(\mathcal{F})$ can be used to reconstruct an optimal solution to $\mathcal{F}$ in polynomial time. The goal of (sound) preprocessing is to increase the applicability of MaxSAT for solving optimization problems by decreasing the overall time spent solving instances. In other words, effective preprocessing makes the total time spent preprocessing $\mathcal{F}(p)$ together with the time spent solving $pre(\mathcal{F}(p))$ lower than the time required to directly solve $\mathcal{F}(p)$. In this thesis, we focus on problem-independent preprocessing, i.e., preprocessing that does not depend on the particular optimization problem $\mathcal{P}$ being solved. In other words, we focus on preprocessing techniques that can be applied on any MaxSAT instance $\mathcal{F}$, regardless of the particular domain from which $\mathcal{F}$ was obtained.

In SAT solving, the importance of preprocessing is well-understood [140]. Many modern SAT solvers apply preprocessing before starting search [141–150]. The effectiveness of preprocessing in SAT solving suggests that similar effective preprocessing rules could be developed for MaxSAT solving as well. This possibility is especially interesting in the context of SAT-based MaxSAT solvers, since their effectiveness relies heavily on SAT solvers. Generalizing preprocessing rules proposed for SAT solving to MaxSAT is not straightforward. Direct application of many such rules to MaxSAT instances is not sound [135]. Informally, the reason is that, in order to be sound for SAT-solving, a preprocessing rule should preserve satisfiability, not the number of falsified clauses, and thus not optimal MaxSAT solutions either [151].

One approach to sound MaxSAT preprocessing is the so-called MaxSAT resolution rule [151]. Preprocessing rules based on MaxSAT resolution are indeed used by some B&B solvers [131, 152, 153]. However, such rules are difficult to use efficiently when solving MaxSAT instances that correspond to industrial applications. The reason is that each application of MaxSAT resolution adds several new clauses to the instance. Hence, MaxSAT resolution based preprocessing rules often increase the size of the already large industrial instances beyond what MaxSAT solvers can handle. In this thesis we focus on an alternative approach to MaxSAT preprocessing known as label-based preprocessing [135, 154]. Label-based preprocessing of MaxSAT instances allows generalizing several of the existing and well-established preprocessing rules proposed for SAT solving to MaxSAT by adding a single new variable (a label) to each soft clause of the instance before preprocessing.

In Papers I and II we develop label-based preprocessing further. In Paper I we show how label-based preprocessing can efficiently be incorporated with SAT-based MaxSAT solvers. The central insight of Paper I is that most SAT-based MaxSAT solvers add extra variables to the soft clauses of MaxSAT instances regardless of the use of preprocessing. In Paper I we show that the labels added during preprocessing can be reused in the solver, thus avoiding the need for the solver to add any new variables. We also show that reusing variables improves the empirical performance of LMHS, an at the time state-of-the-art SAT-based MaxSAT solver [123]. In Paper II we take the idea further and show that some variables in the input MaxSAT instance itself can be reused in the preprocessing and solving phases. This further reduces the number of new variables that need to be added when preprocessing and solving MaxSAT instances. We also show that identifying reusable variables from MaxSAT instances improves empirical performance of LMHS.

In Paper III we present a theoretical analysis on the effect of preprocessing on the number of SAT solver calls that SAT-based MaxSAT solvers require in order to terminate. An underlying motivation for the analysis is that SAT solver calls are the most computationally expensive step of such solvers. Thus insights into which factors influence the number of necessary calls can potentially significantly improve them. In Paper III we show that label-based generalizations of preprocessing rules for SAT solving can not reduce the minimum number of necessary SAT solver calls. We also show that preprocessing can ensure that the solver avoids worst-case executions, i.e., that preprocessing can decrease the maximum number of iterations required by SAT-based MaxSAT solvers.

Finally, in Paper IV we propose and analyze a new label-based MaxSAT preprocessing rule called subsumed label elimination (SLE). We analyze the theoretical differences between SLE and the generalizations of preprocessing rules for SAT solving. In particular, we show that including SLE amongst the preprocessing rules used during label-based preprocessing can result in more clauses and variables removed from the instance. We also report on an empirical evaluation on the effect of using SLE during label-based preprocessing. Our results show that SLE can improve the empirical performance of some state-of-the-art SAT-based MaxSAT solvers.

Research Question 2: Applications of MaxSAT in Data Analysis

The second part of this thesis focuses on developing MaxSAT encodings of other NP-hard combinatorial optimization problems. More specifically, Papers V and VI develop new MaxSAT encodings for two data analysis tasks: correlation clustering [136] (Paper V) and bounded treewidth Bayesian network structure learning [137, 155] (Paper VI).

Clustering is one of the central problems of unsupervised machine learning [156–159]. Given a set of data points, the goal of clustering is to partition the set in some meaningful way. The partitioning is typically called a clustering of the data and each set of a clustering is a cluster. This definition of clustering is very general, a number of different clustering problems and algorithms have been proposed over the years [160–164], including some constraint-based approaches [8–10, 165–167]. In Paper V, we focus on the correlation clustering problem [168–174]. Correlation clustering is a recently proposed clustering paradigm geared towards classifying data based on qualitative similarity information—as opposed to quantitative information—of pairs of data points. An instance of the correlation clustering problem consists of a set of data points and pairwise similarity information over them. The similarity information expresses preferences on whether or not the pair of points should be assigned to the same cluster. Informally, pairs of points that are similar should be assigned to the same cluster. At the same time, pairs of points that are dissimilar should be assigned to different clusters. An optimal solution to the instance balances these two conflicting objectives as well as possible. In contrast to other typical clustering paradigms, correlation clustering does not require the number of clusters as input. Instead, the optimal number should be learned during search. This makes correlation clustering especially well-suited for settings in which the true number of clusters is unknown. Consider for example the problem of clustering documents by topic without any prior knowledge on what those topics might be or how many of them there are [136, 175].

In Paper V we propose and prove the correctness of three MaxSAT encodings of correlation clustering. We also empirically compare the resulting MaxSAT-based solution approach with previously proposed exact and approximation (inexact) algorithms. Our results indicate that, within the scalability of exact approaches, the MaxSAT-based approach is often both faster and more memory-efficient than other exact approaches. We also show that the clusterings obtained using MaxSAT are of significantly better quality than the ones obtained by inexact algorithms, especially on sparse instances with missing similarity information.

Bayesian networks are an important class of probabilistic graphical models widely-used for representing joint probability distributions of sets of random variables [137, 176]. A Bayesian network structure is a directed acyclic graph (DAG) in which each node corresponds to a random variable. The graph represents the conditional dependencies between the variables. Often, a Bayesian network structure that represents given data well is not known *a priori*, and needs to be learned from observations (data) instead. Learning the optimal structure is a well-known optimization problem called the Bayesian network structure learning problem (BNSL) [177–180]. There are two main frameworks for BNSL: the score-based framework, and the independence test-based framework. In the score-based framework, each possible DAG structure is assigned a score that measures how well the structure explains the observations. The goal of BNSL is to compute a best-scoring network. For several commonly used scoring functions, the BNSL problem is NP-hard [181]. As is typical for challenging optimization problems, early solution methods to the problem tended to focus on polynomial-time inexact algorithms [182–187] while interest in exact algorithms for BNSL has increased within the last decade [54, 180, 188–191].

After having learnt a Bayesian network structure, the network is typically used for *probabilistic inference* tasks, such as inferring the probability distribution of some variables, possibly given the values of others. For general Bayesian network structures, this inference task is NP-hard [192]. However, it is becomes tractable whenever the underlying network structure has *bounded (fixed) treewidth* [193, 194]. Treewidth is a well-known graph-theoretic measure [195]. Informally, treewidth measures how “close” a given graph is to being a tree. All trees have treewidth 1 and all complete graphs with n nodes have treewidth $n-1$. Treewidth has important connections to (in)tractability. Many NP-hard problems become tractable when restricted to instances that can be modeled using graphs with bounded treewidth [196, 197]. The fact that inference is tractable in Bayesian networks with low treewidth motivates the development of algorithms that learn optimal Bayesian network structures with bounded treewidth, a problem known as bounded treewidth Bayesian network structure learning (BTBNSL). Compared to the recent progress in practical algorithms for optimally solving BNSL, fewer algorithms have been proposed for BTBNSL [138, 198–201]. In Paper VI we study BTBNSL in the score-based framework. It should be noted that the extra constraint bounding the treewidth of the solution network structure is a non-trivial addition to BNSL. BTBNSL is also an NP-hard optimization problem [199]. In fact, computing the treewidth of any graph is NP-hard [202].

In Paper VI we propose a MaxSAT encoding of BTBNSL. We compare the resulting MaxSAT-based solution approach to a previously proposed dynamic programming algorithm as the only other practical exact solution algorithm to BTBNSL available at the time of the publication of Paper VI [198]. We show that the MaxSAT-based method is more memory-efficient and scales noticeably better than the dynamic programming algorithm.

1.2.3 Specific Contributions by the Present Author

All publications were jointly co-written by all of their authors. Other contributions by the present author are as follows.

Paper I: The idea of reusing labels in a SAT-based MaxSAT solver was first proposed by the present author. The modifications required for label reusing in LMHS were done by the second author of the paper as the author of the LMHS solver. The present author modified a SAT preprocessor to be usable as a MaxSAT preprocessor and ran all of the experiments.

Paper II: The idea of identifying literals from the input formula that can be used as labels in preprocessing and assumptions in MaxSAT solving was a natural extension of Paper I. The present author implemented the modifications to the external preprocessor used in the publication and ran all of the experiments.

Paper III: The theoretical analysis was conducted by the present author under the guidance of the second author.

Paper IV: The idea of subsumed label elimination was developed by the present author with assistance from the other authors. The present author implemented the technique into the preprocessor and ran all of the experiments.

Paper V: The MaxSAT encoding of correlation clustering was jointly developed by the authors of the publication. The present author ran all of the experiments.

Paper VI: The MaxSAT encoding of bounded treewidth Bayesian network learning was co-developed by the authors of the publication. The present author ran all experiments presented in the paper.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2 we give the background information relevant to this thesis. The contributions to the first and second research question are then overviewed in more detail in Chapters 3 and 4, respectively. We conclude the thesis with a summarizing discussion in Chapter 5.

Chapter 2

Preliminaries

In this chapter we give the relevant definitions and background information for understanding the main results of this thesis. First we give precise definitions of the satisfiability and maximum satisfiability problems in Sections 2.1 and 2.2, respectively. We then proceed by overviewing *cardinality constraints* in Section 2.3 as an important class of higher level constraints commonly used in both SAT-based MaxSAT solving and MaxSAT encodings of other optimization problems. We end the chapter by overviewing SAT-based MaxSAT solvers in Section 2.4. In our discussion we assume familiarity with propositional logic.

2.1 Propositional Satisfiability

We identify the truth value true with 1 and false with 0. A Boolean variable x has the domain $\{0, 1\}$. A literal l is a Boolean variable x or its negation $\neg x$. For a literal l, it holds that $\neg\neg l = l$. A clause C is a disjunction ($\vee$) of literals and a formula in conjunctive normal form (CNF) is a conjunction ($\wedge$) of clauses. We will mostly treat clauses as sets of literals and CNF formulas as sets of clauses. We will also simplify set notation when modifying formulas. Specifically, given a clause C and a CNF formula F, $F \setminus C$ is identified with $F \setminus \{C\}$ and $F \cup C$ with $F \cup \{C\}$. We denote the set of variables and literals of a clause C by $\textsc{Var}(C)$ and $\textsc{Lit}(C)$, respectively. The set of variables $\textsc{Var}(F)$ and literals $\textsc{Lit}(F)$ of a formula F are defined by $\textsc{Var}(F) = \bigcup_{C \in F} \textsc{Var}(C)$ and $\textsc{Lit}(F) = \bigcup_{C \in F} \textsc{Lit}(C)$, respectively. For a set L of literals, we use $\neg L$ to denote the set of negations of the literals in L, i.e., $\neg L = \{\neg l \mid l \in L\}$. L is a set of *assumptions* if either $x \notin L$ or $\neg x \notin L$ for each variable $x \in \textsc{Var}(F)$. Given a literal l, we denote by $\textsc{Cl}_F(l)$ the set of clauses of F which contain l, dropping the subscript whenever clear from context. The clauses $\textsc{Cl}(L)$ containing literals from the set $L \subseteq \textsc{Lit}(F)$ are defined by $\textsc{Cl}(L) = \cup_{l \in L}\textsc{Cl}(l)$.

Given a set V of Boolean variables, a truth assignment τ over V is a function $\tau\colon V \to \{0,1\}$. A truth assignment is extended to literals, clauses and CNF formulas in the standard way: $\neg x$ is true ($\tau(\neg x) = 1$) if x is false ($\tau(x) = 0$), a clause C is true ($\tau(C) = 1$) if $\tau(l) = 1$ for at least one literal $l \in C$, and a CNF formula F is true ($\tau(F) = 1$) if $\tau(C) = 1$ for all clauses $C \in F$. A truth assignment τ satisfies a clause C if $\tau(C) = 1$ and a formula if $\tau(F) = 1$, else it falsifies them. A CNF formula F is satisfiable if there exists a truth assignment τ which satisfies it, else F is unsatisfiable. Two formulas F_1 and F_2 are *equivalent* if $\tau(F_1) = \tau(F_2)$ for any truth assignment τ over $\textsc{Var}(F_1) \cup \textsc{Var}(F_2)$. The formulas are *equisatisfiable* if F_1 is satisfiable if and only if F_2 is. The well-known propositional satisfiability (SAT) problem asks if a given CNF formula F is satisfiable. As is common in most practical applications, we treat the SAT problem as the problem of computing a satisfying assignment to F or proving that one does not exist. Essentially all modern SAT solvers can provide a satisfying assignment whenever invoked on a satisfiable formula F.

A truth assignment $\tau\colon S \to \{0,1\}$ over a proper subset $S \subset \textsc{Var}(F)$ is a partial assignment of the formula F. The simplification of F under a partial truth assignment τ is another formula F^{τ} obtained by removing all clauses satisfied by τ from the formula and all literals falsified by τ from the remaining clauses. When convenient, we will treat a (partial) truth assignment τ as a set of literals by $l \in \tau$ if and only if $\tau(l) = 1$. Similarly, each set $L \subseteq \textsc{Lit}(F)$ of assumptions can be treated as a (partial) truth assignment. In this thesis we use partial truth assignments in the context of satisfiability checking under assumptions [203]. Given a formula F and a set of assumptions $L \subseteq \textsc{Lit}(F)$, we say that F is satisfiable under L if F^L is satisfiable. For an alternative view, F is satisfiable under L if there exists a satisfying assignment τ to F that sets $\tau(l) = 1$ for all $l \in L$.

Given a CNF formula F, a *SAT solver* is an algorithm that computes a satisfying assignment to F or proves that one does not exist. The development of SAT solvers is an active area of research [73, 74, 204–210]. Besides pure satisfiability checking, SAT solvers are commonly used as subroutines in more complex algorithms, for example in SAT-based MaxSAT solvers [116, 203]. Most modern SAT solvers that are used in SAT-based MaxSAT solving implement the conflict-driven clause learning (CDCL) algorithm [204, 211, 212]. CDCL solvers have in turn evolved from the older Davis-Putnam-Logemann-Long search procedure [213]. In this thesis we only use CDCL SAT solvers as black boxes in SAT-based MaxSAT solvers and as such will not discuss the details of how they operate here. The only requirement we make of a SAT solver is that it supports satisfiability querying under assumptions, and that it is able to compute subsets of the assumptions which explain unsatisfiability. More precisely, given a formula F and a set of assumptions $L \subseteq \textsc{Lit}(F)$ for which F^L is unsatisfiable, we assume that the SAT solver can extract a subset $L' \subseteq L$ such that $F^{L'}$ is unsatisfiable as well. Most modern CDCL SAT solvers support these features through the so-called *assumption interface.*

2.2 Maximum Satisfiability

An instance $\mathcal{F}$ of weighted partial maximum satisfiability (or MaxSAT for short) is a triplet $\mathcal{F} = (F_h, F_s, w)$ consisting of two CNF formulas, the *hard clauses* F_h and the *soft clauses* F_s, and a weight function $w\colon F_s \to \mathbb{N}$. The literals $\textsc{Lit}(\mathcal{F})$ and variables $\textsc{Var}(\mathcal{F})$ of MaxSAT instances are the literals and variables of $F_h \wedge F_s$, respectively. Given a MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, any truth assignment τ which satisfies F_h is a solution to $\mathcal{F}$. The cost $\mathrm{COST}(\mathcal{F}, \tau)$ of a solution τ to $\mathcal{F}$ is the sum of the weights of soft clauses it falsifies, i.e.,

$$\mathrm{COST}(\mathcal{F}, \tau) = \sum_{C \in F_s} w(C) \cdot (1 - \tau(C)).$$

A solution τ^o to $\mathcal{F}$ is optimal if $\mathrm{COST}(\mathcal{F}, \tau^o) \leq \mathrm{COST}(\mathcal{F}, \tau)$ for all solutions τ to $\mathcal{F}$. The (optimal) cost of the instance $\mathcal{F}$ is the cost of the optimal solutions to $\mathcal{F}$. We denote the optimal cost of an instance $\mathcal{F}$ by $\mathrm{COST}(\mathcal{F})$. In the rest of the thesis, we assume that all MaxSAT instances have solutions, or equivalently, that F_h is satisfiable.

The MaxSAT solvers we work with in this thesis make extensive use *unsatisfiable cores.* Given a MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, a subset $\kappa \subseteq F_s$ is an unsatisfiable core if the formula $F_h \wedge \kappa$ is unsatisfiable. A core κ is a minimal if $F_h \wedge \kappa_s$ is satisfiable for all $\kappa_s \subset \kappa$. Minimal cores are abbreviated by MUS (minimal unsatisfiable subformula). A set $M \subseteq F_s$ is a correction set (of $\mathcal{F}$) if the formula $F_h \wedge (F_s \setminus M)$ is satisfiable. The correction set M is minimal (an MCS) if $F_h \wedge (F_s \setminus M_s)$ is unsatisfiable for all $M_s \subset M$. We denote the set of MUSes and MCSes of $\mathcal{F}$ by $\mathrm{MUS}(\mathcal{F})$ and $\mathrm{MCS}(\mathcal{F})$, respectively.

The MCSes and MUSes of MaxSAT instances are related to each other via *hitting sets.* Given a collection of sets $\mathcal{K}$, a set H is a hitting set over $\mathcal{K}$ if $H \cap K \neq \emptyset$ for all $K \in \mathcal{K}$. A hitting set H is *irreducible* if no $H_s \subset H$ is a hitting set over $\mathcal{K}$. For a MaxSAT instance $\mathcal{F}$, the well known hitting set duality theorem establishes a connection between $\mathrm{MUS}(\mathcal{F})$ and $\mathrm{MCS}(\mathcal{F})$.

Theorem 1 (Hitting Set Duality [214])**.** *A set κ is an MUS of a MaxSAT instance $\mathcal{F}$ if and only if it is an irreducible hitting set over* $\mathrm{MCS}(\mathcal{F})$. *Similarly, a set M is an MCS of $\mathcal{F}$ if and only if it is an irreducible hitting set over* $\mathrm{MUS}(\mathcal{F})$.

Minimal correction sets provide an alternative definition of the MaxSAT problem. For a solution τ to a MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, let $\mathrm{U}(\tau) \subseteq F_s$ be the set of soft clauses falsified by τ. We say that τ is a *minimal solution* to $\mathcal{F}$ if $\mathrm{U}(\tau)$ is set-minimal, i.e., if there does not exist a solution τ^2 to $\mathcal{F}$ for which $\mathrm{U}(\tau^2) \subset \mathrm{U}(\tau)$. Notice that all optimal solutions to $\mathcal{F}$ are minimal but the converse does not hold. It is simple to show that there exists a many-to-one correspondence between minimal solutions and the MCSes of $\mathcal{F}$. More specifically, a solution τ of $\mathcal{F}$ is minimal if and only if $\mathrm{U}(\tau) \in \mathrm{MCS}(\mathcal{F})$. We say that a minimal solution τ to $\mathcal{F}$ corresponds to an MCS M^τ of $\mathcal{F}$ if $M^\tau = \mathrm{U}(\tau)$. The correspondence is not one-to-one, instead each $M \in \mathrm{MCS}(\mathcal{F})$ corresponds to a set of minimal solutions of $\mathcal{F}$. However, if two minimal solutions τ^1 and τ^2 to $\mathcal{F}$ correspond to the same $M \in \mathrm{MCS}(\mathcal{F})$, then τ^1 and τ^2 satisfy (and hence also falsify) the exact same clauses of $\mathcal{F}$. This implies that

$$\textsc{Cost}(\mathcal{F}, \tau^1) = \sum_{\substack{C \in F_s \\ \tau^1(C)=0}} w(C) = \sum_{C \in M} w(C) = \sum_{\substack{C \in F_s \\ \tau^2(C)=0}} w(C) = \textsc{Cost}(\mathcal{F}, \tau^2).$$

In this thesis we will treat minimal solutions that correspond to the same MCSes as equivalent. We say that an $M \in \mathrm{MCS}(\mathcal{F})$ corresponds to a solution τ^M if τ^M is a minimal solution of $\mathcal{F}$ that corresponds to M. A set $M \in \mathrm{MCS}(\mathcal{F})$ is optimal if it corresponds to an optimal solution of $\mathcal{F}$.

The relationship between MCSes and minimal solutions of MaxSAT instances suggests an alternative definition of the MaxSAT problem. Let $\mathcal{F} = (F_h, F_s, w)$ be a MaxSAT instance with the weight function w extended to sets $S \subseteq F_s$ of soft clauses by $w(S) = \sum_{C \in S} w(C)$. Denote the set of solutions and minimal solutions to $\mathcal{F}$ by $\textsc{sol}(\mathcal{F})$ and $\textsc{msol}(\mathcal{F})$, respectively. The optimal cost $\mathrm{COST}(\mathcal{F})$ of $\mathcal{F}$ can be expressed in terms of the MCSes of $\mathcal{F}$ by

$$\begin{aligned}\mathrm{COST}(\mathcal{F}) &= \min_{\tau \in \textsc{sol}(\mathcal{F})} \mathrm{COST}(\mathcal{F}, \tau) = \min_{\tau \in \textsc{msol}(\mathcal{F})} \mathrm{COST}(\mathcal{F}, \tau) \\ &= \min_{\tau \in \textsc{msol}(\mathcal{F})} w(M^\tau) = \min_{M \in \mathrm{MCS}(\mathcal{F})} w(M).\end{aligned}$$

In other words, an $M \in \mathrm{MCS}(\mathcal{F})$ is optimal if $w(M) \leq w(M')$ for all $M' \in \mathrm{MCS}(\mathcal{F})$. Thus the MaxSAT problem can be reformulated as the problem of computing an $M^o \in \arg\min_{M\in \text{MCS}(\mathcal{F})}\{w(M)\}$. By hitting set duality, such M^o is also a *minimum-cost hitting set* over $\text{MUS}(\mathcal{F})$, i.e., a hitting set over $\text{MUS}(\mathcal{F})$ which minimizes $w(M^o)$ over all hitting sets of $\text{MUS}(\mathcal{F})$. Notice that a minimum-cost hitting set is guaranteed to be irreducible. The following theorem shows that a satisfiability query can be used in order to verify that a hitting set over any collection of cores of $\mathcal{F}$ is an optimal MCS without computing the entire $\text{MUS}(\mathcal{F})$.

Theorem 2 (Adapted from [122]). *Let $\mathcal{F} = (F_h, F_s, w)$ be a MaxSAT instance and $\mathcal{C}$ a collection of cores of $\mathcal{F}$. Let M be a minimum cost hitting set over $\mathcal{C}$ and assume that $F_h \wedge (F_s \setminus M)$ is satisfiable. Then M is an optimal MCS of $\mathcal{F}$.*

The implicit hitting set solvers we work with in this thesis are based on Theorem 2.

2.3 Cardinality Constraints

Despite the simple syntax, several types of complex constraints can be modeled with CNF formulas. One such class of constraints commonly used in both SAT-based MaxSAT solving and MaxSAT encodings of other problems are *cardinality constraints*, an important special case of the more general class of *pseudo-boolean constraints*. Given a set $L = \{l_1, \ldots, l_n\}$ of n literals, a set $W = \{w_1, \ldots, w_n\}$ of weights, a constant k and $\circ \in \{\leq, \leq, \geq\geq, =\}$, a pseudo-boolean constraint is a linear constraint over L of form $\sum_{i=1}^{n} w_i l_i \circ k$. A truth assignment τ satisfies the constraint whenever $\sum_{i=1}^{n} w_i \tau(l_i) \circ k$ is true. We denote the set of clauses resulting from encoding a pseudo-boolean constraint $\sum_{i=1}^{n} w_i l_i \circ k$ to CNF by $\text{CNF}(\sum_{i=1}^{n} w_i l_i \circ k)$. A pseudo-boolean constraint is a cardinality constraint if $w_i = 1$ for all $1 \leq i \leq n$. The numerous applications of cardinality constraints have motivated the development several different CNF encodings of them [215–220].

Example 1. *Let $L = \{l_1, \ldots, l_N\}$ be a set of literals and consider the* at-most-one *cardinality constraint*

$$\sum_{i=1}^{N} l_i \leq 1$$

enforcing that at most one of the literals in L must be set to true. The at-most-one constraint is commonly used in SAT-based MaxSAT solving [111, 115, 116, 120] as well as MaxSAT encodings of other problems, including correlation clustering and bounded treewidth Bayesian network structure *learning. A simple way of encoding this constraint in CNF is with $\mathcal{O}(n^2)$ clauses of form $(\neg l_i \vee \neg l_j)$ for every distinct l_i and l_j in L. As an example of a more compact encoding, the ladder encoding uses $n-1$ auxiliary variables $y_1, \ldots, y_{n-1}$ and clauses corresponding to $l_i \leftrightarrow (\neg y_i \wedge y_{i+1})$ as well as $y_i \rightarrow y_{i+1}$. All in all the ladder encoding uses $\mathcal{O}(n)$ auxiliary variables and $\mathcal{O}(n)$ clauses.*

2.4 SAT-based MaxSAT Solvers

In this section we overview and discuss the two types of MaxSAT solvers on which the rest of the thesis focuses on. The contributions of this thesis to MaxSAT preprocessing are not specific to a single MaxSAT solver, but instead two classes of MaxSAT solvers that we call core-guided solvers [111, 115–117, 119, 120] and implicit hitting set based solvers [121–123]. We discuss these solvers in terms of two abstract MaxSAT solving algorithms: *CG*, representing core-guided solvers and *IHS*, representing implicit hitting set based solvers. In the rest of the thesis, we use the term MaxSAT algorithm to refer to abstractions and MaxSAT solver to refer to concrete implementations of MaxSAT algorithms.

The CG and IHS algorithms are presented in pseudocode in Figure 2.1 on the left and right side, respectively. These abstractions cover several modern MaxSAT solvers, including Fu-Malik (WPM1, WMSU1) [116, 221, 222], PMRES [115], OLL [111, 223] and ONE (K) [119] (the CG algorithm), as well as MaxHS [121, 122] and LMHS [123] (the IHS algorithm). It should be noted that solvers implementing CG or IHS often also make use of several different additional heuristics and search strategies [224–227] that are not included in the pseudocodes of Figure 2.1.

Both CG and IHS rely extensively on the ability to extract unsatisfiable cores from MaxSAT instances. Let (F_h, F_s) be two sets of clauses such that $F_h \wedge F_s$ is unsatisfiable. In both CG and IHS, a core $\kappa \subseteq F_s$ is extracted using the assumption interface of the underlying SAT solver. Let $F_s^A = \{C \vee a_C \mid C \in F_s\}$ be the set of all clauses in F_s, each extended with a unique *assumption variable* a_C. Let also $\mathcal{A}(F_s) = \textsc{Var}(F_s^A) \setminus \textsc{Var}(F_s)$ be the set of all assumption variables and consider a subset $\mathcal{A}_s \subseteq \mathcal{A}(F_s)$. Core extraction using assumptions is based on the fact that the simplification of $F_h \wedge F_s^A$ under $\neg\mathcal{A}_s$ is the formula $F_h \wedge \{C \mid C \vee a_C \in \textsc{Cl}(\mathcal{A}_s)\}$. In order to see this, consider a clause $C \vee a_C \in F_s^A$. If $a_C \in \mathcal{A}_s$, the partial assignment $\neg\mathcal{A}_s$ reduces $C \vee a_C$ to C. If $a_C \notin \mathcal{A}_s$, the clause $C \vee a_C$ can be trivially satisfied by setting a_C to true. Hence we can check the satisfiability of $F_h \wedge F_s$ by querying a SAT solver for the satisfiability of $F_h \wedge F_s^A$ under $\neg\mathcal{A}(F_s)$.

```
1  CG($F_h, F_s, w$)
2  ($F_h^w, F_s^w$) ← ($F_h, F_s$)
3  while true do
4      ($result$, $\kappa, \tau$) ← IsSAT($F_h^w, F_s^w$)
5      if $result$="satisfiable" then
6          return $\tau$
7      else
8          $F_s^w = (F_s^w \setminus \kappa)$
9          $F_s^w \leftarrow F_s^w \wedge$ CLONE($\kappa$)
10         ($F_h^w, F_s^w$) ← RELAX($F_h^w, F_s^w, \kappa$)
```

```
1  IHS($F_h, F_s, w$)
2  $\mathcal{C} \leftarrow \emptyset$
3  while true do
4      $H \leftarrow$ MinCostHittingSet($\mathcal{C}$)
5      ($result$, $\kappa, \tau$) ← IsSAT($F_h, (F_s \setminus H)$)
6      if $result$="satisfiable" then
7          return $\tau$
8      else
9          $\mathcal{C} \leftarrow \mathcal{C} \cup \{\kappa\}$
```

Figure 2.1: Abstractions of the two types of MaxSAT algorithms we work with in this thesis.

If the formula is satisfiable, the returned truth assignment (restricted to $\textsc{Var}(F_h \wedge F_s)$) is also a satisfying assignment of $F_h \wedge F_s$. Otherwise, the subset $\neg\mathcal{A}_\kappa \subseteq \neg\mathcal{A}(F_s)$ of the assumptions returned by the solver can be mapped to an unsatisfiable core $\kappa = \{C \mid C \vee a_C \in \textsc{Cl}(\mathcal{A}_\kappa)\}$ of (F_h, F_s). In Figure 2.1 we abstract this functionality into the function IsSAT. The result of a query IsSAT(F_h, F_s) is a triplet (*result*, κ, τ), where *result* is true if and only if $F_h \wedge F_s$ is satisfiable. If $F_h \wedge F_s$ is satisfiable, then τ is a satisfying assignment to it. Otherwise κ is an unsatisfiable core of $F_h \wedge F_s$. In the IHS algorithm the assumption interface is also used for removing clauses from the SAT-solver queries. More specifically, for a subset $H \subseteq F_s$, the satisfiability of $F_h \wedge (F_s \setminus H)$ is equivalent to the satisfiability of $F_h \wedge F_s^A$ under $\neg\mathcal{R}^H = \neg(\mathcal{A}(F_s) \setminus \mathcal{A}(H))$. This enables clause removal from the formula without the need to reset the internal state of the SAT solver. Notice that if $F_h \wedge F_s^A$ is unsatisfiable under $\neg\mathcal{R}^H$, the core returned by the SAT solver is guaranteed to be a subset of $F_s \setminus H$.

Given an input MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, the CG algorithm maintains a working formula (F_h^w, F_s^w), initialized to (F_h, F_s) on Line 2. The algorithm iteratively queries the internal SAT solver using the function IsSAT(F_h^w, F_s^w) (Line 4), obtaining a triplet (*result*, κ, τ). Whenever the SAT solver returns "satisfiable", CG terminates and returns the assignment τ, guaranteed to be an optimal solution to $\mathcal{F}$ (Line 6). Otherwise, a core κ of (F_h^w, F_s^w) is obtained. The algorithm proceeds by *relaxing* the working instance and *compiling* information about the core into it (Line 10). Most of the implementations of RELAX that we are aware of assume that all of the soft clauses in the core have equal weight. To handle cores κ with varying clause weights, the solvers use a standard technique known as clause cloning [109, 221] (Line 9). First the smallest weight among all clauses in κ is computed, $w^\kappa_{\min} = \min\{w(C) \mid C \in \kappa\}$. Then each clause $C \in \kappa$ for which $w(C) > w^\kappa_{\min}$ is cloned; a duplicate clause $\textsc{Clone}(C)$ is added to F_s^w, the weight of the original clause is set to $w_{\min}^{\kappa}$, and the duplicate $\textsc{Clone}(C)$ is given the residual weight $w(\textsc{Clone}(C)) = w(C) - w_{\min}^{\kappa}$. All duplicates are left in the working instance as soft clauses and the function $\text{RELAX}(F_h^w, F_s^w, \kappa)$ is invoked using the original clauses of κ which now all have equal weight. The exact manner in which the formula is modified, i.e., the implementation of RELAX, depends on the concrete MaxSAT solver. A classical example is the Fu-Malik solver [116] in which each clause $C \in \kappa$ is extended with a fresh relaxation variable r_C to form the extended clause $C \vee r_C$. The extended clauses are left in the formula as soft and a cardinality constraint $\text{CNF}(\sum r = 1)$ is added as hard clauses. Several of the early core-guided solvers relax the soft clauses in the core and add cardinality constraints as hard clauses. In contrast, more recently proposed core-guided solvers harden the soft clauses in the core and add cardinality constraints as soft clauses [111, 115, 119, 120].

In contrast to the CG algorithm, the IHS algorithm does not add or remove any clauses at all during execution and instead only works on the input hard and soft clauses. Given an input MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, the IHS algorithm maintains a set $\mathcal{C}$ of cores of $\mathcal{F}$, initialized to $\emptyset$ on Line 2. At each iteration, a minimum-cost hitting set over $\mathcal{C}$ is computed (Line 4). Then a SAT solver is invoked on all of the clauses in the working formula, except for the ones in H (Line 5). If the formula is satisfiable, H is an optimal MCS of $\mathcal{F}$ (Theorem 2) and IHS terminates, returning the optimal solution satisfying $F_h \wedge (F_s \setminus H)$ (Line 7). Otherwise, a new core is obtained and added to the set $\mathcal{C}$ (Line 9), after which the algorithm reiterates. In two solvers implementing the IHS algorithm, MaxHS [122] and LMHS [123], a minimum-cost hitting set is obtained by solving the current hitting set problem using an integer programming solver.

Beyond the scope of this thesis, a third class of SAT-based MaxSAT solvers are the so-called model-guided solvers [103–108, 110]. When invoked on a MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, a model-guided solver initializes an upper and lower bound UB and LB of the optimal cost of $\textsc{Cost}(\mathcal{F})$. The exact manner in which the bounds are initialized depends on the solver, a simple example sets $\text{LB} = 0$ and $\text{UB} = \sum_{C \in F_s} w(C)$. During search, the solver queries a SAT solver for the satisfiability of $F_h \wedge F_s^A \wedge \text{CNF}(\sum_{C \in F_s}(w(C) \cdot a_C) \leq k)$ where k is some constant satisfying $LB \leq k \leq UB$. If the formula is satisfiable, then $\textsc{Cost}(\mathcal{F}) \leq k$ and the value of the upper bound is lowered. Similarly, if the formula is unsatisfiable, then $\textsc{Cost}(\mathcal{F}) > k$ and the value of the lower bound is increased. The solver iterates until an optimal solution is found. Different model-guided solvers make use of several different search strategies and encodings of cardinality constraints. Some also use unsatisfiable cores for more precise control on which soft clauses are relaxed and how much the bounds are updated [106, 107].

Finally we mention that in addition to SAT-based MaxSAT solvers a central approach to exact MaxSAT solving is branch and bound (B&B) [124–133]. Given an input MaxSAT instance $\mathcal{F}$, B&B solvers search for an optimal solution to $\mathcal{F}$ by branching on the two possible values of each variable in the formula. In order to avoid exhaustive search over all possible assignments of the variables, B&B solvers make use of several different bound computation and other inference rules [131, 134, 152, 228, 229] designed to allow effective pruning of the search tree. Some B&B solvers also make use of restricted forms of unsatisfiable cores in their bound computations [125].

Chapter 3

Preprocessing for Maximum Satisfiability Solving

In this chapter we discuss the contributions of this thesis to MaxSAT preprocessing techniques, overviewing Papers I-IV. While the importance of preprocessing in SAT solving is well-established [140–144, 146–150], the role of preprocessing in MaxSAT solving is not as developed [135, 151]. Here we focus on the label-based approach to MaxSAT preprocessing [135] and the CG and IHS MaxSAT algorithms presented in Figure 2.1 of Section 2.4. The empirical results presented in this chapter focus on the LMHS MaxSAT solver [123], a from-scratch instantiation of the IHS algorithm by the second author of Papers I, II and IV. All experiments were performed on a cluster of 2.53-GHz Intel Xeon quad core machines with 32 GB memory and Ubuntu Linux, using a per-instance memory limit of 30 GB. Since the time limit used in the experiments varied between papers, we will specify them in the relevant sections. For the formal proofs and complete empirical results, we direct the reader to the reprints of the papers at the end of the thesis.

This chapter is organized as follows. In Section 3.1 we give preliminaries on label-based preprocessing of MaxSAT instances. In Section 3.2 we discuss how label-based preprocessing can be integrated into SAT-based MaxSAT solving in a manner that significantly decreases the number extra variables and clauses that are added (Papers I and II). We demonstrate that tighter integration between the preprocessing and solving steps results in improved empirical performance of LMHS. In Section 3.3 we overview a theoretical analysis of the effect of preprocessing on the number of iterations required by CG and IHS (Paper III). Finally, in Section 3.4 we present a MaxSAT-specific preprocessing technique that we call subsumed label elimination (SLE) (Paper IV). We give theoretical results on the differences between SLE and the MaxSAT generalizations of preprocessing rules for SAT solving. We also show that using SLE in conjunction with previously proposed preprocessing rules leads to further simplifications during preprocessing as well as improved empirical performance of LMHS.

3.1 Label-based MaxSAT Preprocessing

Most of the contributions of this thesis to MaxSAT preprocessing build on previous work [135] on lifting four central preprocessing rules proposed for SAT to MaxSAT using the so called labeled CNF (LCNF) framework [154, 230]. More specifically, the rules lifted are bounded variable elimination, subsumption and self-subsuming resolution [141], as well as blocked clause elimination [231]. In this chapter we focus on the same four rules and call them *SAT-based preprocessing rules.* It should, however, be noted that, in addition to these four, several other preprocessing rules have been proposed for SAT solving [140, 142, 143, 145–150].

For some intuition on why SAT-based preprocessing rules can not directly be applied on MaxSAT instances, consider the subsumption elimination (SE) rule. Let F be a SAT formula and C, D two clauses of F. We say that C subsumes D if $C \subseteq D$. A clause D is *subsumed* if some other clause subsumes it. The SE rule allows removing subsumed clauses from F. Let $pre(F)$ be the formula resulting after an application of SE on F. Then F and $pre(F)$ are equisatisfiable since any assignment τ that satisfies the former will satisfy the latter and vice versa. More generally, we say that a preprocessing rule is *sound* for SAT-solving if (i) applying the rule to a formula F gives an equisatisfiable formula $pre(F)$ and (ii) a satisfying assignment to F can be reconstructed from any satisfying assignment to $pre(F)$ in polynomial time. Even if SE is sound for SAT solving, the next example demonstrates that directly removing subsumed clauses from the hard and soft clauses of MaxSAT instances can alter the costs of solutions and thus also the optimal solutions.

Example 2. *Let $\mathcal{F} = (F_h, F_s, w)$ be a MaxSAT instance with*

$$\begin{aligned} F_h &= \{(\neg x_1), (\neg x_2), (\neg x_3 \vee \neg x_4)\}, \\ F_s &= \{(x_1 \vee x_3), (x_2 \vee x_3), (x_3), (x_4)\} \end{aligned}$$

and $w(C) = 1$ for each $C \in F_s$. An optimal solution τ to $\mathcal{F}$ sets $\tau(x_1) = \tau(x_2) = \tau(x_4) = 0$ and $\tau(x_3) = 1$, falsifying one soft clause. Direct application of SE on $F_h \wedge F_s$ removes two soft clauses. The resulting instance

$\mathcal{F}^2 = (F_h^2, F_s^2, w^2)$ has

$$F_h^2 = \{(\neg x_1), (\neg x_2), (\neg x_3 \lor \neg x_4)\} \text{ and } F_s^2 = \{(x_3), (x_4)\}.$$

One optimal solution τ^2 to $\mathcal{F}^2$ sets $\tau^2(x_1) = \tau^2(x_2) = \tau^2(x_3) = 0$ and $\tau^2(x_4) = 1$. This solution falsifies one soft clause in $\mathcal{F}^2$ but three in $\mathcal{F}$.

Example 2 illustrates the fact that instead of only preserving satisfying assignments, MaxSAT preprocessing should preserve the *optimal solutions* of instances.

Definition 1. *Let $\mathcal{F}$ be a MaxSAT instance, $\mathcal{R}$ a preprocessing rule, and $pre(\mathcal{F})$ the instance obtained by preprocessing $\mathcal{F}$ with $\mathcal{R}$. Assume τ^p is an optimal solution to $pre(\mathcal{F})$. The preprocessing rule $\mathcal{R}$ is sound for MaxSAT if an optimal solution τ to $\mathcal{F}$ can be reconstructed from τ^p in polynomial time.*

Procedure 3.1 describes label-based preprocessing of a MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$ using SAT-based preprocessing rules. First, each soft clause $C \in F_s$ is extended with a unique new *label* (Boolean variable) l_C to form the *labeled clause* $C \lor l_C$ and the set of labeled soft clauses $F_s^L = \{C \lor l_C \mid C \in F_s\}$. Notice the similarity between labels and assumption variables used in SAT-based MaxSAT solving (recall Section 2.4). Let $\mathcal{L}(\mathcal{F}) = \textsc{Var}(F_s^L) \setminus \textsc{Var}(F_s)$ be the set of all added labels. The next step of label-based preprocessing is preprocessing the formula $F_h \land F_s^L$ with SAT-based preprocessing rules, thereby obtaining the formula $pre(F_h \land F_s^L)$. In order to guarantee soundness for MaxSAT, bounded variable elimination and self-subsuming resolution are restricted from removing any variables in $\mathcal{L}(\mathcal{F})$ during preprocessing [135]. Finally, the preprocessed MaxSAT instance $pre(\mathcal{F}) = (F_h^p, F_s^p, w^p)$ has $F_h^p = pre(F_h \land F_s^L)$ as hard clauses. The soft clauses F_s^p contain unit clauses with negations of labels that appear among the hard clauses: $F_s^p = \{(\neg l_C) \mid l_C \in \textsc{Lit}(F_h^p) \cap \mathcal{L}(\mathcal{F})\}$. The weight of each soft clause $w^L((\neg l_C))$ is equal to the weight $w(C)$ of the soft clause to which l_C was added in the first step. We emphasize that soft clauses are only added for labels $l_C \in \textsc{Lit}(F_h^p)$. Even if bounded variable elimination or self subsuming resolution can not remove any labels, a label can still be removed from the instance during preprocessing. For example, if a soft clause is subsumed by a hard clause, SE can remove the labeled soft clause during preprocessing together with the corresponding label.

The basis for the soundness of label-based preprocessing with SAT-based preprocessing rules is the following theorem.

Preprocess $\mathcal{F} = (F_h, F_s, w)$

1. Let $F_s^L = \{C \lor l_C \mid C \in F_s, l_C \text{ new}\}$ and $\mathcal{L}(\mathcal{F}) = \textsc{Var}(F_s^L) \setminus \textsc{Var}(F_s)$.
2. Preprocess the CNF formula $F_h \land F_s^L$ using SAT-based preprocessing rules.
   - Do not remove any $l \in \mathcal{L}(\mathcal{F})$ with bounded variable elimination nor self-subsuming resolution.
3. Return
   $$pre(\mathcal{F}) = (F_h^p, F_s^p, w^p)$$
   with $F_h^p = pre(F_h \land F_s^L)$, $F_s^p = \{(\neg l_C) \mid l_C \in \textsc{Lit}(F_h^p) \cap \mathcal{L}(\mathcal{F})\}$ and $w^p((\neg l_C)) = w(C)$.

Procedure 3.1: Label-based preprocessing of a MaxSAT instance $\mathcal{F}$.

Theorem 3. *(Adapted from [135]) Assume that an instance $\mathcal{F} = (F_h, F_s, w)$ is preprocessed using label-based preprocessing with SAT-based preprocessing rules to obtain $pre(\mathcal{F}) = (F_h^p, F_s^p, w^p)$. For each soft clause $C \in F_s$, let l_C be the label added to C during preprocessing. Then the following hold.*

(i) The optimal costs of $\mathcal{F}$ and $pre(\mathcal{F})$ are equal.

(ii) $M \in \text{MUS}(\mathcal{F})$ if and only if $\{(\neg l_C) \mid C \in M\} \in \text{MUS}(pre(\mathcal{F}))$.

We show that label-based preprocessing with SAT-based preprocessing rules is sound for MaxSAT using Theorem 3 and hitting set duality (Theorem 1).

Theorem 4. *Label-based preprocessing with SAT-based preprocessing rules is sound for MaxSAT.*

Proof. (Sketch) Let $\mathcal{F} = (F_h, F_s, w)$ be a MaxSAT instance and $pre(\mathcal{F}) = (F_h^p, F_s^p, w^p)$ an instance obtained by label-based preprocessing of $\mathcal{F}$ using SAT-based preprocessing rules. Consider an optimal solution τ^p to $pre(\mathcal{F})$ and let M^{τ^p} be the MCS corresponding to τ^p. By Theorem 3 and hitting set duality, the set $M = \{C \mid (\neg l_C) \in M^{\tau^p}\}$ is an MCS of $\mathcal{F}$. Since $w^p((\neg l_C)) = w(C)$ for all $(\neg l_C) \in M^{\tau^p}$, it follows that $w^p(M^{\tau^p}) = w(M)$. Since M^{τ^p} is optimal for $pre(\mathcal{F})$ and $\text{COST}(pre(\mathcal{F})) = \text{COST}(\mathcal{F})$, the

MCS M is optimal for $\mathcal{F}$. Hence the solution τ^M corresponding to M is an optimal solution to $\mathcal{F}$. The fact that τ^M can be reconstructed from τ^p in polynomial time follows from the fact that SAT-based preprocessing rules allow reconstruction of satisfying assignments to CNF formulas [135, 140]. $\square$

3.2 Integrating Label-based Preprocessing into SAT-based Solving

As the first contribution to MaxSAT preprocessing of this thesis we investigate label-based preprocessing in conjunction with SAT-based MaxSAT solving. In Paper I we show how to improve the empirical performance of LMHS [123], a MaxSAT solver implementing the IHS algorithm, by reusing labels as assumptions.

More generally, we show that if a preprocessed MaxSAT instance $pre(\mathcal{F})$ is solved with a SAT-based MaxSAT solver, the labels introduced during preprocessing can be reused as the assumption variables used for core extraction within the internal SAT solver. Since a similar number of assumption variables would otherwise be introduced by the solver, reusing labels as assumptions removes the need to add extra variables when using label-based preprocessing with SAT-based MaxSAT solving. In more detail, assume that the IHS algorithm instructed to reuse labels as assumptions is invoked on a preprocessed MaxSAT instance $pre(\mathcal{F}) = (F_h^p, F_s^p, w^p)$. Then the internal SAT solver of IHS is first initialized with the clauses in F_h^p (and specifically not F_s^p). During search, the cores in $\mathcal{C}$ are maintained in terms of label variables. Each computed hitting set H is the set of label variables that should not be assumed to be false in the next SAT solver call. Each unsatisfiable SAT solver call obtains a new subset of the label variables and the augmented IHS algorithm terminates as soon as a SAT solver call returns "satisfiable". While Paper I focuses on the IHS algorithm, a similar idea is applicable to the CG algorithm as well. Notice that labels l can be treated as soft clauses by introducing a unit clause $(\neg l)$ on demand.

Informally, the correctness of reusing labels as assumptions follows by considering an execution of IHS not reusing labels as assumptions invoked on a preprocessed MaxSAT instance $pre(\mathcal{F}) = (F_h, F_s, w)$. Initially each soft clause $(\neg l_C) \in F_s$ is extended with an assumption variable a to form the clause $(\neg l_C \vee a)$. Notice that this clause is equivalent to the implication $\neg a \rightarrow \neg l_C$. Let $\mathcal{A}(pre(\mathcal{F}))$ be the set of all assumption variables. During an iteration of the while-loop (Lines 3-9), IHS will first compute H as a minimum-cost hitting set over the set of cores $\mathcal{C}$ identified so far. In practice, H is a subset of $\mathcal{A}(pre(\mathcal{F}))$ containing the assumption variables a of all clauses $\neg l_C \vee a$ which will be removed from the instance in the next SAT solver call. Afterwards a SAT solver is invoked on $F_h \wedge F_s^A$ under $\neg(\mathcal{A}(pre(\mathcal{F})) \setminus H)$. In this call, each soft clause $(\neg l_C \vee a)$ for which $a \notin H$ is reduced to $(\neg l_C)$ due to assuming a to be false. Thus the value of l_C is propagated to false as well. Similarly, if $a \in H$, the value of a is not assumed to be anything at all. However, as a only appears in a single clause, the SAT solver can assign it to true in order to satisfy the clause $(\neg l_C \vee a)$. As $\neg l_C$ does not appear in any other clause, the SAT solver can also assign l_C to true, satisfying all clauses in $\textsc{Cl}_{F_h}(l_C)$. Thus the assumptions only affect the values of the corresponding label variables through the implications $\neg a \rightarrow \neg l_C$. An alternative description of reusing labels as assumptions is hence to not introduce the implications $\neg a \rightarrow \neg l_C$ at all, but instead directly assume the values of the l_C variables. In Paper I we give a more direct proof of soundness using the formal LCNF framework [135].

In addition to the theoretical analysis, Paper I also reports on an experimental evaluation of the effect that reusing labels as assumptions has on LMHS. The evaluation was performed using the weighted partial industrial and crafted benchmarks of the 2014 MaxSAT evaluation [101] using a per-instance time limit of 1 h. Figure 3.2 shows a summary of the results. The line MaxHS-2.5 corresponds to the newest version of the MaxHS solver at the time [121, 122]. MaxHS is included to give a baseline comparison to LMHS.

The line LMHS+pre of Figure 3.2 of shows the performance of LMHS using preprocessing *without reusing* labels as assumptions. We note that preprocessing without reusing labels actually degrades overall performance of the solver. The best overall performance is achieved by LMHS+R-pre, corresponding to LMHS using preprocessing and reusing labels as assumptions. In the rest of this chapter, we will refer to LMHS+R-pre simply as LMHS, explicitly mentioning whenever it is used *without* reusing labels as assumptions.

Reusing Literals from MaxSAT Instances

In Paper II we show how the number of variables that need to be introduced to a MaxSAT instance $\mathcal{F}$ during preprocessing and SAT-based solving can be decreased further. We prove that literals $l \in \textsc{Lit}(\mathcal{F})$ that satisfy three easily identifiable criteria can be reused as labels in preprocessing and assumptions in SAT-based MaxSAT solving. We also propose *group detection* as a simple pattern-matching procedure to identify such literals. We experimentally demonstrate that reusable literals can be iden-

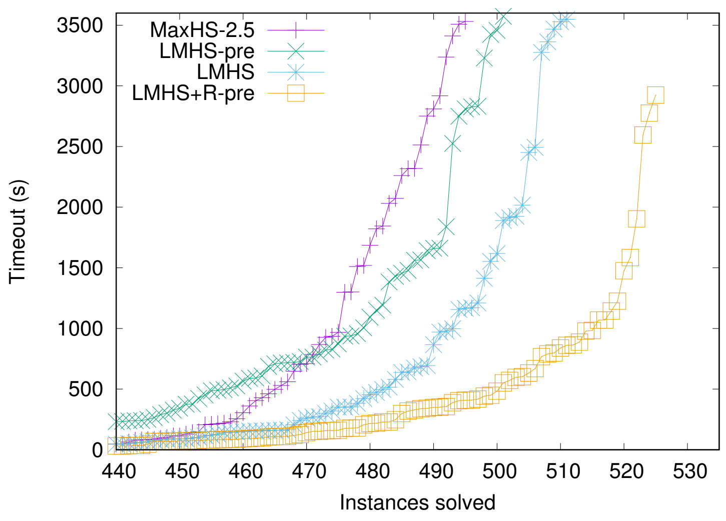

Figure 3.2: The effect of reusing labels as assumptions on LMHS (from Paper I).

tified in a significant fraction of the MaxSAT evaluation benchmarks and that using group detection leads to modest improvements in the empirical performance of LMHS.

In more detail, we introduce the concept of a *group-detectable* literal.

Definition 2. *Let $\mathcal{F} = (F_h, F_s, w)$ be a MaxSAT instance and $l \in \textsc{Lit}(\mathcal{F})$. The literal l is* group-detectable *if it satisfies the following three criteria.*

1. $(\neg l) \in F_s$.
2. $\neg l \notin \textsc{Lit}(F_h \wedge (F_s \setminus (\neg l)))$.
3. $l \notin \textsc{Lit}(F_s)$.

In words, a literal l is group-detectable in an instance $\mathcal{F} = (F_h, F_s, w)$ if l is not a member of any soft clauses of $\mathcal{F}$ and its negation $\neg l$ does not appear in any clause in $F_h \wedge F_s$ except for one unit soft clause. We say that a soft clause $(\neg l)$ is group-detectable if the literal l is.

In Paper II we show that given *any* MaxSAT instance $\mathcal{F}$ (preprocessed or not), all group-detectable literals $l \in \textsc{Lit}(\mathcal{F})$ can be reused as labels for preprocessing and assumptions for core extraction. For some intuition on the connection between Paper I and II, notice that if an instance $\mathcal{F}$ is preprocessed to obtain a preprocessed instance $pre(\mathcal{F})$, then every soft clause in $pre(\mathcal{F})$ is group-detectable. Hence group detection could be seen as a generalization of reusing labels as assumptions after preprocessing.

In Paper II we propose group detection as a pattern matching procedure for identifying group-detectable literals and reusing them as labels during preprocessing and assumptions during solving. Intuitively, the correctness of group detection should be clear. In the paper we give a formal proof of correctness using the LCNF framework [135]. The name group detection stems from a setting in which we are given a group G of clauses and wish to encode the soft group constraint $\bigwedge_{C \in G} C$ of weight w_g in CNF. One possible encoding is to: (i) introduce a single new *group variable* g, (ii) extend each $C \in G$ with the same g variable to form the clause $C \vee g$, (iii) treat all extended clauses $C \vee g$ as hard, and (iv) introduce the soft clause $(\neg g)$ with weight c_w. Notice that using this encoding the literal g is group-detectable. An observation similar to Paper II was made in [232] where the authors study *group MaxSAT* as an alternative approach to handling soft group constraints.

In Paper II we report on the results of an empirical evaluation of group detection. Figure 3.3 shows the fraction of soft clauses group-detectable in the weighted partial industrial and crafted benchmarks of the 2014 MaxSAT evaluation. As the figure illustrates, all soft clauses are group-detectable in over 40% of the crafted and over 30% of the industrial instances, suggesting that a significant fraction of the soft clauses in the considered MaxSAT benchmarks correspond to encodings of group constraints. The effect of group detection on the total solving time of LMHS is shown in Figure 3.4 on the same benchmark set. All runs were performed using a per-instance time limit of 1 h. The base algorithm (the line LMHS in the plot), using neither preprocessing nor group detection, exhibits the worst performance. Interestingly, the variant using only group detection and no preprocessing (LMHS-G) is competitive with the variant using preprocessing without group detection (LMHS-pre). This observation highlights the importance of minimizing the number of extra variables and clauses during label-based preprocessing and SAT-based MaxSAT solving. Best performance is achieved by using both preprocessing and group detection (LMHS-G-pre), even if the improvement over LMHS-G and LMHS-pre is modest. In Paper II we offer one possible explanation for the modesty of the improvement to be the fairly strong connection between group detection and bounded variable elimination. Apart from the LMHS solver, the paper also includes empirical results for the Eva solver [115] as an example of a solver that implements the CG algorithm.

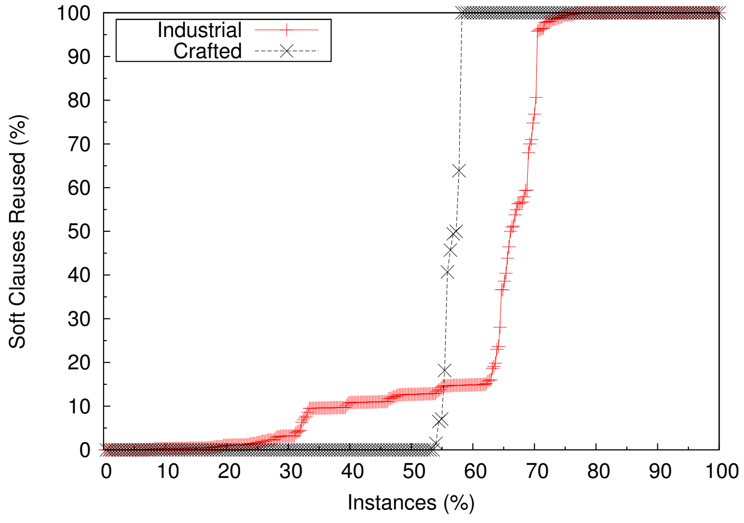

Figure 3.3: The fraction of soft clauses reusable as labels or assumptions (from Paper II).

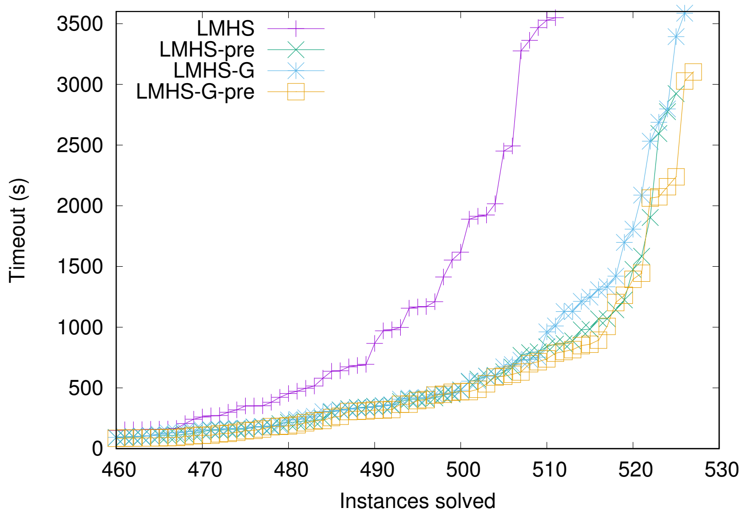

Figure 3.4: The effect of group detection on the LMHS MaxSAT solver (from Paper II).

3.3 Effect of Preprocessing on SAT-based MaxSAT Solving

Paper III focuses on improving the theoretical understanding of the effect of label-based preprocessing on the number of SAT solver calls made by SAT-based MaxSAT algorithms. We note that theoretical analysis of many SAT-based MaxSAT algorithms is in general challenging. For example, the number of SAT solver calls necessary for the CG algorithm to solve any MaxSAT instance is not known. The difficulty in determining the number stems to some extent from the fact that the instance is modified during search. Thus the core extracted on each iteration is a core of the current instance but not necessarily a core of the original instance. More generally, the effect that the formula rewriting step of the CG algorithm has on the core structure of the input instance remains an interesting open question, even if some results are known [233]. Analysis of the IHS algorithm is simpler since it only extracts cores of the original instance. It is known that the number of SAT solver calls necessary for the IHS algorithm to solve MaxSAT instances can be exponential in the number of soft clauses, even when restricted to unweighted instances [234].

In Paper III we provide a full characterization of the effect of label-based preprocessing with SAT-based preprocessing rules on the number of necessary SAT solver calls of two algorithms: IHS and CG^H. CG^H is an abstract MaxSAT algorithm first studied in [233]. In particular CG^H is similar to CG except for the fact that CG^H only considers implementations of Relax in which the soft clauses of the core remain in the instance as *soft clauses* and new cardinality constraints are added as *hard clauses*. In [233] the authors provide a characterization of the cores in the ith working formula of CG^H in terms of the cores of the input instance and the added (hard) cardinality constraints. Our results in Paper III for CG^H make use of this characterization. As a by-product of the main result, we also develop a similar characterization of the MUSes, thus sharpening the main results of [233].

In order to simplify the discussion, we will from now on focus on what we call *normalized MaxSAT instances*, a view on MaxSAT instances similar to [119]. A MaxSAT instance $\mathcal{F}^N = (F_h^N, F_s^N, w)$ is normalized if each soft clause $C \in F_s^N$ is group-detectable. We say that a literal $l \in \text{Lit}(\mathcal{F}^N)$ is a *soft literal* if $(\neg l) \in F_s^N$. The results of Papers I and II imply that we can assume all MaxSAT instances to be normalized. In more detail, given any MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$ we construct a normalized instance $\mathcal{F}^N$ by replacing each $C \in F_s$ which is not group detectable by a hard clause

$C \lor g_C$ and a soft clause $(\neg g_C)$ of weight $w(C)$. The results of Papers I and II imply that $\mathcal{F}^N$ has the same optimal solutions as $\mathcal{F}$ and reusing all the group-detectable literals of $\mathcal{F}^N$ as assumptions allows solving $\mathcal{F}^N$ without introducing any extra variables compared to solving $\mathcal{F}$. Normalized MaxSAT instances are convenient for the analysis conducted in Paper III. If a normalized MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$ is preprocessed with group detection to obtain the instance $pre(\mathcal{F}) = (F_h^p, F_s^p, w)$, then $F_s^p \subseteq F_s$, and Theorem 3 can be restated as follows.

Corollary 1. *Let $\mathcal{F}$ be a normalized MaxSAT instance and $pre(\mathcal{F})$ the instance resulting after preprocessing $\mathcal{F}$ using label-based preprocessing with group detection and SAT-based preprocessing rules. Then* $\mathrm{MUS}(\mathcal{F}) = \mathrm{MUS}(pre(\mathcal{F}))$*, which implies* $\mathrm{MCS}(\mathcal{F}) = \mathrm{MCS}(pre(\mathcal{F}))$.

While Paper III focuses on SAT-based preprocessing rules, the results hold for any preprocessing rules that satisfy Corollary 1.

In Paper III we analyze four variants of a fixed $\mathrm{A} \in \{\mathrm{CG}^H, \mathrm{IHS}\}$.

- A: the base algorithm.
- A_{pre}: A applied after label-based preprocessing.
- $\mathrm{A}^{\mathrm{MUS}}$: A using an idealized SAT solver that is guaranteed to extract an MUS when invoked on an unsatisfiable formula.
- $\mathrm{A}^{\mathrm{MUS}}_{pre}$: $\mathrm{A}^{\mathrm{MUS}}$ applied after label-based preprocessing.

We investigate the relative performance of these variants from two separate points of view, the *best case* and *worst case* in the number of iterations (SAT solver calls). Let $\mathcal{F}$ be a normalized MaxSAT instance and $\mathrm{A} \in \{\mathrm{CG}^H, \mathrm{IHS}\}$. Due to non-deterministic heuristics of SAT solvers, there are several different possible executions of A invoked on $\mathcal{F}$. We define the *length* of an execution of A on $\mathcal{F}$ as the number of cores extracted by A during that execution before termination. A *best-case* execution of A on $\mathcal{F}$ has length equal to the minimum over all possible executions of A on $\mathcal{F}$. Similarly, a *worst-case* execution has length equal to the maximum over all possible executions. Let $\textsc{Minlen}(\mathrm{A}, \mathcal{F})$ and $\textsc{Maxlen}(\mathrm{A}, \mathcal{F})$ denote the length of best-case and worst-case executions of A on $\mathcal{F}$, respectively.

Figures 3.5 and 3.6 overview the results of our analysis. We establish that for any $\mathrm{A} \in \{\mathrm{CG}^H, \mathrm{IHS}\}$, label-based preprocessing using SAT-based preprocessing rules can not decrease the length of the best-case executions of A on any instance (Figure 3.5). On the other hand, preprocessing can decrease the length of the worst-case executions on some instances. Intuitively, the results follow from the inability of SAT-based preprocessing

$$\begin{tikzcd}
\boxed{\mathrm{A}} \arrow[r, bend left] \arrow[d, bend left] & \boxed{\mathrm{A}_{pre}} \arrow[l, bend left] \arrow[d, bend left] \\
\boxed{\mathrm{A}^{\mathrm{MUS}}} \arrow[u, bend left] \arrow[r, bend left] & \boxed{\mathrm{A}^{\mathrm{MUS}}_{pre}} \arrow[l, bend left] \arrow[u, bend left]
\end{tikzcd}$$

Figure 3.5: Best-case performance in the number of iterations of A $\in \{\mathrm{CG}^H, \mathrm{IHS}\}$. Here $X \to Y$ indicates that $\textsc{Minlen}(X, \mathcal{F}) \leq \textsc{Minlen}(Y, \mathcal{F})$ on all MaxSAT instances $\mathcal{F}$ (from Paper III).

$$\begin{tikzcd}
\boxed{\mathrm{A}} \arrow[r, bend left, "/" marking] \arrow[d, bend left, "\backslash" marking] & \boxed{\mathrm{A}_{pre}} \arrow[l, bend left] \arrow[d, bend left, "\backslash" marking] \\
\boxed{\mathrm{A}^{\mathrm{MUS}}} \arrow[u, bend left] \arrow[r, bend left] & \boxed{\mathrm{A}^{\mathrm{MUS}}_{pre}} \arrow[l, bend left] \arrow[u, bend left]
\end{tikzcd}$$

Figure 3.6: Worst-case performance in the number of iterations of A $\in \{\mathrm{CG}^H, \mathrm{IHS}\}$. Here $X \to Y$ indicates that $\textsc{Maxlen}(X, \mathcal{F}) \leq \textsc{Maxlen}(Y, \mathcal{F})$ on all MaxSAT instances $\mathcal{F}$. $X \nrightarrow Y$ indicates that $X \to Y$ does not hold (from Paper III).

rules to affect the MUS structure of MaxSAT instances, as stated in Corollary 1. In essence, we show that the best-case executions of A on any instance $\mathcal{F}$ correspond to executions where the SAT solver only returns MUSes. Since SAT-based preprocessing can not affect the MUSes of $\mathcal{F}$, any executions, where the SAT solver only returns MUSes, are valid for both A and A_{pre}. However, as preprocessing can still remove some of the soft literals of MaxSAT instances, applying SAT-based preprocessing before solving might in some cases allow the algorithm to avoid bad executions following from extracting cores that are not MUSes.

Finally, we note that, in addition to CG^H, IHS and the results of Paper III, it is known that $\mathcal{O}(\log(\mathrm{COST}(\mathcal{F})))$ calls to a SAT solver are required to solve any MaxSAT instance [235]. The result is obtained using a model-guided MaxSAT solver which uses a combination of linear and binary search for the optimal cost (recall Section 2.4). As preprocessing does not affect the optimal cost of instances, the same result implies that label-based preprocessing with SAT-based preprocessing rules can not affect the minimum number of SAT solver calls required by the model-guided solver studied in [235] either.

The focus in this section was on the number of iterations of SAT-based solvers. Even if preprocessing does not decrease the minimum number of iterations, the empirical evaluations conducted in Papers I and II demonstrate that preprocessing does improve the empirical performance of SAT-based MaxSAT solvers on some instances. While a complete theoretical understanding of how preprocessing affects SAT-based MaxSAT solvers remains an interesting open research question, we note that the analysis conducted in Paper III does not cover the effect of preprocessing on the individual SAT-solver calls executed by IHS and CG^H.

3.4 Subsumed Label Elimination

As the final contribution of this thesis to MaxSAT preprocessing, we investigate a MaxSAT-specific preprocessing rule which does not satisfy Corollary 1, but still guarantees sound MaxSAT preprocessing. More specifically, in Paper IV we propose and analyze subsumed label elimination (SLE). We prove correctness of SLE, give theoretical analysis comparing SLE to SAT-based preprocessing rules, and show empirically that incorporating SLE in the preprocessing step further improves performance of several MaxSAT solvers implementing CG and IHS. In the paper we also briefly overview the connection between SLE and the so-called *column dominance rule* proposed in the early 90s in conjunction with branch-and-bound approaches for the *binate covering problem* [236]. We will not detail the binate covering problem here, except to say that while SLE can be seen as the MaxSAT counterpart of the column dominance rule, Paper III reports on the first study of such a rule in the context of MaxSAT that we are aware of.

Similarly to the previous section, we assume that all MaxSAT instances in this section are normalized. This allows identifying MCSes with sets of soft literals. More specifically, let $\mathcal{F} = (F_h, F_s, w)$ be a normalized MaxSAT instance and consider a subset $K \subseteq F_s$ of soft clauses. We are interested in determining the satisfiability of the formula $F_K = F_h \wedge (F_s \setminus K)$. Let $C \in K$ and $D \in (F_s \setminus K)$ be two soft clauses of $\mathcal{F}$. Then $C = (\neg l_1)$ and $D = (\neg l_2)$ for two soft literals l_1 and l_2. Now $\neg l_1 \notin \textsc{Lit}(F_K)$ since C is the only clause of $F_h \wedge F_s$ containing $\neg l_1$. Thus all clauses in $\textsc{Cl}_{F_K}(l_1)$ can be trivially satisfied by assigning l_1 to true. On the other hand, as $D \in F_K$, any potential satisfying assignment τ to F_K assigns l_2 to false. In other words, any potential satisfying assignment to F_K can set all soft literals l for which $(\neg l) \in K$ to true and has to set all literals l for which $(\neg l) \in F_s \setminus K$ to false. Hence the satisfiability of F_K is equivalent to the satisfiability of the simplification $F_h^{K^L}$ of F_h under the partial assignment $K^L = \{l \mid (\neg l) \in K\} \cup \{\neg l \mid (\neg l) \in F_s \setminus K\}$. Thus K is a correction set of $\mathcal{F}$ if and only if $F_h^{K^L}$ is satisfiable. In this section, we identify sets $M \subseteq F_s$ of soft clauses of normalized MaxSAT instances with the partial assignment $M^L = \{l \mid (\neg l) \in M\} \cup \{\neg l \mid (\neg l) \in F_s \setminus M\}$. Specifically, we define the simplification F_h^M of F_h under $M \subseteq F_s$ to be equal to $F_h^{M^L}$.

Next we give an informal description of SLE. Let $\mathcal{F} = (F_h, F_s, w)$ be a normalized MaxSAT instance, and l_1 and l_2 two soft literals of $\mathcal{F}$. We say that l_2 subsumes l_1 if (i) $\textsc{Cl}(l_1) \subseteq \textsc{Cl}(l_2)$, i.e., l_2 appears in all of the same clauses as l_1, and (ii) $w((\neg l_1)) \geq w((\neg l_2))$. SLE allows removing subsumed literals from $\mathcal{F}$. More formally, SLE allows enforcing all subsumed literals to false and simplifying the instance accordingly. The soundness of SLE follows from the fact that if l_1 is subsumed by l_2, then there exists an optimal $M \in \text{MCS}(\mathcal{F})$ which does not contain $(\neg l_1)$. Hence there also exists an optimal solution τ^M to $\mathcal{F}$ that assigns l_1 to false. More specifically, we show that if $(\neg l_1)$ is a member of some $M^1 \in \text{MCS}(\mathcal{F})$, then $M^2 = (M^1 \setminus (\neg l_1)) \cup (\neg l_2)$ is a correction set of $\mathcal{F}$. Thereby M^2 contains an MCS M_s of $\mathcal{F}$ for which $(\neg l_1) \notin M_s$. Notice that the assumption $w((\neg l_1)) \geq w((\neg l_2))$ implies that $w(M^1) \geq w(M^2) \geq w(M_s)$, so either M^1 is not optimal, or all three are. To see that M^2 is a correction set, notice that the simplifications of F_h under M^1 and M^2 satisfy $F_h^{M^2} \subseteq F_h^{M^1}$ and recall that $F_h^{M^1}$ is satisfiable[1].

[1]After publication of Paper IV we have discovered a minor error in the proof of the

In Paper IV we provide a formal proof of correctness of SLE on the LCNF level. In more detail, for any normalized MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$, we show that the soft literal l_1 is subsumed by the soft literal l_2 if (i) $(\neg l_2)$ appears in the same MUSes of $\mathcal{F}$ as $(\neg l_1)$ and (ii) $w((\neg l_1)) \geq w((\neg l_2))$. However, as checking which literals belong to which MUSes is NP-hard, the first condition is probably not checkable in polynomial time. Instead, we show that $\textsc{Cl}(l_1) \subseteq \textsc{Cl}(l_2)$ is a sufficient condition for $(\neg l_2)$ appearing in the same MUSes as $(\neg l_1)$.

In addition to its proof of correctness, Paper IV also contains additional theoretical analysis of SLE. We show that, in contrast to the SAT-based preprocessing rules, SLE does not satisfy Corollary 1. Instead, we show that the MUSes of MaxSAT instances $pre(\mathcal{F})$ preprocessed with SLE are restrictions of the MUSes of $\mathcal{F}$ onto the soft clauses of $pre(\mathcal{F})$. The contrast between SLE and SAT-based preprocessing is further exemplified in Paper IV by MaxSAT instances on which no SAT-based preprocessing rules can be applied but SLE can. The possibility of SLE affecting the MUSes of MaxSAT instances also means that preprocessing with SLE can in some cases remove optimal MaxSAT solutions. More precisely, assume that two labels l_1 and l_2 subsume each other, i.e., that $\textsc{Cl}(l_1) = \textsc{Cl}(l_2)$ and $w((\neg l_1)) = w((\neg l_2))$. Then, if there exists an optimal $M_1 \in \text{MCS}(\mathcal{F})$ containing l_1 and not l_2, there also exists an optimal $M_2 \in \text{MCS}(\mathcal{F})$ containing l_2 and not l_1. Even so, SLE can soundly remove either l_1 or l_2, thus also removing the corresponding MCS and optimal MaxSAT solutions. However, as implied by the soundness of SLE, it will never remove all optimal solutions nor create new ones.

It should be noted that only using SLE during preprocessing might not be very effective. The reason is the precondition $\textsc{Cl}(l_1) \subseteq \textsc{Cl}(l_2)$. In order for this condition to be met, the formula needs to contain clauses with more than one soft literal. However, many normalized MaxSAT instances encountered in practical applications need not contain such clauses. Consider for example an un-normalized MaxSAT instance $\mathcal{F} = (F_h, F_s, w)$ that does not contain any group-detectable soft clauses. Then every clause in the normalized instance $\mathcal{F}^N = (F_h^N, F_s^N, w)$ obtained from $\mathcal{F}$ following Section 3.3 contains at most one soft literal. Even so, SLE can still be used when preprocessing $\mathcal{F}^N$ as long as other techniques capable of distributing soft literals among clauses are used as well. In our work the main preprocessing rule capable of this is bounded variable elimination.

theorem corresponding to this discussion, Theorem 6. The last sentence of the proof should read: "By assumption, $R' = (R \setminus \{l_2\}) \cup \{l_1\}$, a subset of $Lbls(\Phi^{pre})$, is a hitting set of LMUS(Φ) and hence contains an irreducible hitting set of LMUS(Φ), i.e, an LMCS of Φ."

In Paper IV we present the result of an experimental evaluation on the effect of SLE together with SAT-based MaxSAT solvers. As benchmarks we used the industrial and crafted benchmarks of the 2015 MaxSAT evaluation. Figure 3.7 demonstrates the effect that SLE has on the fraction of soft literals remaining after preprocessing with and without SLE. We found that especially on weighted instances, SLE can significantly increase the number of soft literals that are removed during preprocessing. For example, for one third of the weighted partial industrial instances ($x = 0.3$), with SLE close to 80% of the soft literals are eliminated ($y \approx 0.2$, i.e., some 20% of the soft literals remain afterwards). In comparison, without SLE only $\approx 45\%$ are eliminated.

Figure 3.8 gives a break-down of the effect that SLE has on the running time of LMHS on the different families of the weighted partial industrial benchmarks. The experiments were run using a per-instance timeout of 30 min. We see that, for a majority of the instances, SLE improves the total solving time of LMHS, both compared to using no preprocessing, and only using SAT-based preprocessing. In Paper IV, we also provide results for Eva [115], Open-WBO [113], and Primal-Dual [120].

After the publication of Paper IV we have generalized SLE, resulting in a technique called group-subsumed label elimination (GSLE) [237]. A soft literal l of a MaxSAT instance $\mathcal{F}$ is group-subsumed by a set L of soft literals if (i) $\textsc{Cl}(l) \subseteq \textsc{Cl}(L)$ and (ii) $w((\neg l)) \geq \sum_{l_g \in L} w((\neg l_g))$. GSLE allows removing group-subsumed literals from $\mathcal{F}$. GSLE is a straightforward generalization of SLE, the proof of correctness is essentially identical to SLE. Both SLE and GSLE are implemented in the MaxSAT preprocessor MaxPre [237], developed after publication of Paper IV. Interestingly, we have found that using GSLE during preprocessing does not significantly increase total preprocessing time compared to using SLE. In addition to SLE and GSLE, MaxPre also includes all other algorithmic ideas for preprocessing proposed in this thesis, namely label reuse and group detection.

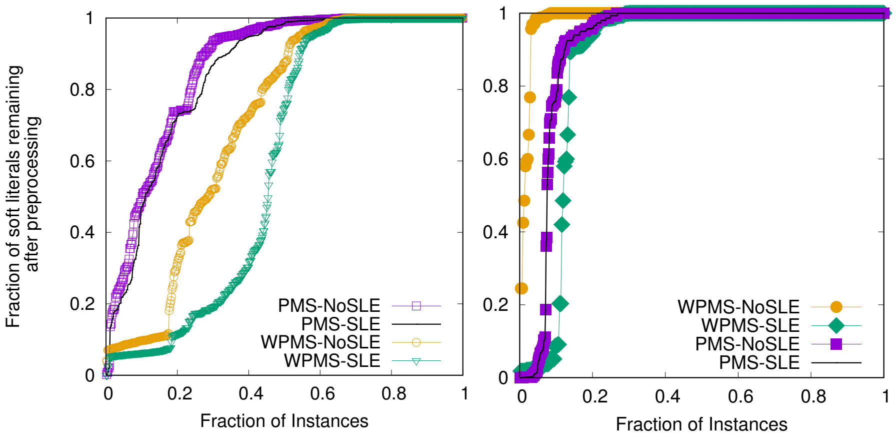

Figure 3.7: Fraction of soft literals remaining in industrial (left) and crafted (right) unweighted (PMS) and weighted (WPMS) benchmarks after preprocessing with and without SLE (from Paper IV).

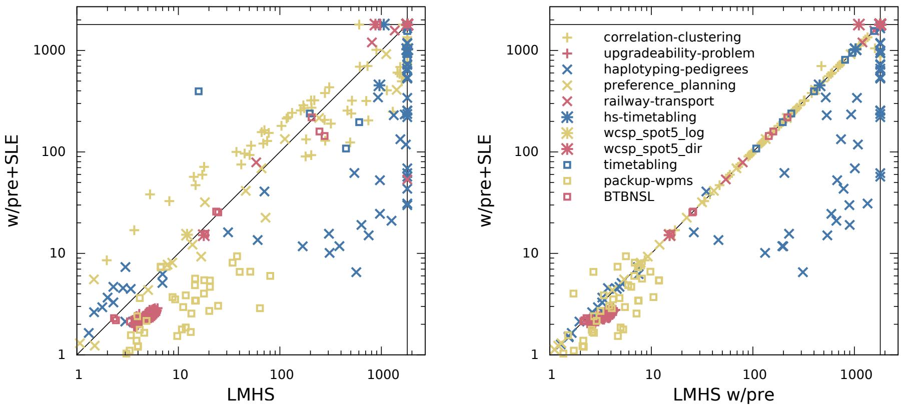

Figure 3.8: Effect of SLE on runtimes without (left) and with (right) SAT-based preprocessing of LMHS on industrial weighted partial instances (from Paper IV).

Chapter 4

Maximum Satisfiability for Data Analysis

In this chapter we discuss the contributions of this thesis to the development of new MaxSAT encodings for two NP-hard data analysis problems, correlation clustering (Paper V) and bounded treewidth Bayesian network structure learning (BTBNSL) (Paper VI). Correlation clustering is discussed in Section 4.1 and BTBNSL in Section 4.2. In both sections we give a precise definition of the data analysis task as a combinatorial optimization problem, overview the MaxSAT encodings we propose, and present a summary of the results of an empirical evaluation of the resulting MaxSAT-based approach.

In order to simplify the discussion, we will present all MaxSAT encodings in this chapter in terms of general propositional logic. This can be done without loss of generality as it is well-known that for any formula $\mathcal{G}$ in propositional logic, there exists an equivalent CNF formula $F^{\mathcal{G}}$. Furthermore, the size of $F^{\mathcal{G}}$ can be assumed to be polynomial in the size of $\mathcal{G}$. More specifically, applying the well-known Tseitin encoding [238] on $\mathcal{G}$ results in an equivalent CNF formula $F^{\mathcal{G}}$ the number of variables and clauses of which are linear in the number of constraints and variables of $\mathcal{G}$.

4.1 Correlation Clustering

In Paper V we present and evaluate three MaxSAT encodings for the correlation clustering problem [239]. Under the original formulation, an instance of correlation clustering consists of an undirected graph with the nodes corresponding to a set of data points and each edge labeled as either positive or negative. Two points with a positive edge between them are similar, and points with a negative edge between them are dissimilar. The goal of correlation clustering is to cluster the nodes of the graph in a way that correlates as well as possible with the edge labeling. More specifically, an optimal clustering balances two conflicting objectives. On one hand, similar points should be assigned to the same cluster. On the other hand, dissimilar points should be assigned to different clusters. Notice that there exists "trivial" clusterings that maximize either individual objective. The number of similar points assigned to the same cluster is maximized by a clustering that assigns all nodes to the same cluster. Similarly, the number of dissimilar points assigned to different clusters is maximized by a clustering that assigns all nodes to different clusters. However, no such trivial clustering is in general optimal with respect to both objectives. Balancing the conflicting objectives is an important characteristic of correlation clustering. The lack of a "trivial" clustering that balances both objectives makes correlation clustering well-suited for situations in which the true number of clusters is unknown.

The rest of the Section is organized as follows. In Section 4.1.1 we detail the general setting under which we study correlation clustering. The three MaxSAT encodings we propose for the problem are presented in Section 4.1.2 and an overview of the results of the experimental evaluation reported on in Paper V is given in Section 4.1.3.

An interest in correlation clustering has continued after the publication of Paper V [240–243]. We especially note a recently published paper that develops one of the MaxSAT encodings that we propose in Paper V in the context of the clique partitioning problem [244].

4.1.1 Problem Setting

An instance of correlation clustering consists of a set $V = \{v_1, \ldots, v_N\}$ of N data points and a symmetric *similarity matrix* $W \in \overline{\mathbb{R}}^{N \times N}$ where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty, -\infty\}$. From now on, we fix V and say that an instance consists only of the matrix W. We denote the element on row i column j in W by $W(i,j)$. The values of W represent similarities of pairs of points, the points v_i and v_j are *similar* if $W(i,j) > 0$ and *dissimilar* if $W(i,j) < 0$. An instance of correlation clustering can equivalently be viewed as an weighted undirected graph $G = (V, E)$ where $\{v_i, v_j\} \in E$ if $W(i,j) \neq 0$, and the weight of each edge $\{v_i, v_j\}$ is equal to $W(i,j)$. Figure 4.1 gives an example of a similarity matrix W on the left and the corresponding graph on the right.

A function $cl\colon V \to \mathbb{N}$ is a clustering of W if $cl(v_i) = cl(v_j)$ for all $W(i,j) = \infty$ and $cl(v_i) \neq cl(v_j)$ for all $W(i,j) = -\infty$. These kinds of

$$W = \begin{bmatrix} \infty & 0 & -2 & 5 \\ 0 & \infty & -\infty & -3 \\ -2 & -\infty & \infty & 1 \\ 5 & -3 & 1 & \infty \end{bmatrix}$$

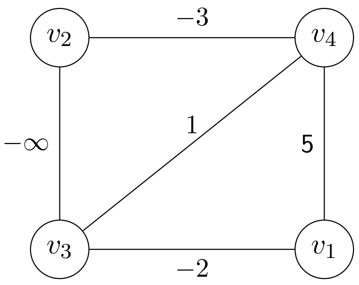

Figure 4.1: A Similarity matrix and its graph representation.

constraints enforcing two points to the same or to different clusters are commonly called *must-link* and *cannot-link* constraints, respectively [245].

Given an instance W of correlation clustering, the cost $\textsc{Cost}(W, cl)$ of a clustering $cl\colon V \to \mathbb{N}$ is

$$\begin{aligned}\textsc{Cost}(W, cl) = &\sum_{\substack{cl(v_i)=cl(v_j)\\ i<j}} (\mathcal{I}[-\infty < W(i,j) < 0] \cdot |W(i,j)|) + \\ &\sum_{\substack{cl(v_i)\neq cl(v_j)\\ i<j}} (\mathcal{I}[\infty > W(i,j) > 0] \cdot W(i,j)),\end{aligned}$$

where $\mathcal{I}[b]$ is an indicator function which takes the value 1 if the condition b is true, else $I[b] = 0$. A clustering cl is optimal if $\textsc{Cost}(W, cl) \leq \textsc{Cost}(W, cl')$ for all clusterings cl' of W.

Example 3. *Consider the similarity matrix W in Figure 4.1 (left). An optimal clustering cl of this instance assigns $cl(v_1) = cl(v_4) = 1$, $cl(v_2) = 2$ and $cl(v_3) = 3$. The cost* $\textsc{Cost}(W, cl)$ *of cl is* 1.

An important factor to note here is that the cost of a clustering does not depend on the specific cluster indexes. Hence the search for an optimal clustering of W can be restricted to functions $cl\colon V \to [N]$ where $[N] = \{0, \ldots, N-1\}$. More generally, given a similarity matrix W, a clustering $cl\colon V \to \mathbb{N}$ and a permutation $\sigma\colon \mathbb{N} \to \mathbb{N}$, the function $cl^\sigma = \sigma \circ cl$ is also a clustering of V with $\textsc{Cost}(W, cl) = \textsc{Cost}(W, cl^\sigma)$. In other words, the space of clusterings is highly symmetric.

4.1.2 MaxSAT Encodings of Correlation Clustering

In this section we overview our three related MaxSAT encodings of correlation clustering, the transitive encoding, the unary encoding and the binary encoding. For the remaining of this section, we fix an instance W of correlation clustering to an $N \times N$ similarity matrix with E non-zero elements. In other words, we assume that there are E pairs of distinct $1 \leq i < j \leq N$ for which $W(i,j) \neq 0$.

For each encoding, we describe the MaxSAT instance $\mathcal{F}(W)$ resulting after applying the encoding on W and a procedure for converting an optimal solution τ of $\mathcal{F}(W)$ to an optimal clustering cl^{τ} of W. In the rest of the section, let $\textsc{Transitive}(W)$, $\textsc{Unary}(W)$ and $\textsc{Binary}(W)$ denote the MaxSAT instances produced by the transitive, unary and binary encodings on W, respectively. Even though the specifics of each instance differ, the overall structure of them is similar. Let $\mathcal{F}(W) \in \{\textsc{Transitive}(W), \textsc{Unary}(W), \textsc{Binary}(W)\}$ and $\mathcal{F}(W) = (F_h^W, F_s^W, w^W)$. The hard clauses F_h^W represent a conjunction of two complex constraints: $F_h^W = \textsc{IsFunc}(W) \wedge \textsc{Sol}(W)$. The constraint $\textsc{IsFunc}(W)$ is satisfied if cl^{τ} is a function $cl^{\tau} : V \to \mathbb{N}$, and $\textsc{Sol}(W)$ is satisfied if cl^{τ} is a clustering of W. The constraint $\textsc{Sol}(W)$ is further divided into two parts corresponding to the must-link ($\textsc{SameCl}(i,j)$) and cannot-link ($\textsc{DiffCl}(i,j)$) constraints, respectively:

$$\textsc{Sol}(W) = \bigwedge_{W(i,j)=\infty} \textsc{SameCl}(i,j) \wedge \bigwedge_{W(i,j)=-\infty} \textsc{DiffCl}(i,j).$$

The constraint $\textsc{SameCl}(i,j)$ is satisfied if and only if $cl^{\tau}(v_i) = cl^{\tau}(v_j)$ and the constraint $\textsc{DiffCl}(i,j)$ is satisfied if and only if $cl^{\tau}(v_i) \neq cl^{\tau}(v_j)$. Since all hard clauses are satisfied by any solution to the MaxSAT instance, the semantics of the constraints ensure that any solution to the MaxSAT instance corresponds to a clustering of W.

The soft clauses F_s^W are defined in a way which ensures that

$$\textsc{Cost}(\mathcal{F}, \tau) = \textsc{Cost}(W, cl^{\tau})$$

for any MaxSAT solution τ. The soft clauses contain $\textsc{SameCl}(i,j)$ with weight $w^W(\textsc{SameCl}(i,j)) = W(i,j)$ for each $0 < W(i,j) < \infty$ and a constraint $\textsc{DiffCl}(i,j)$ with weight $w^W(\textsc{DiffCl}(i,j)) = |W(i,j)|$ for each $-\infty < W(i,j) < 0$; all in all,

$$F_s^W = \bigwedge_{0<W(i,j)<\infty} \textsc{SameCl}(i,j) \wedge \bigwedge_{0>W(i,j)>-\infty} \textsc{DiffCl}(i,j).$$

In Paper V, the correctness of each MaxSAT encoding is established by showing that $\textsc{Cost}(\mathcal{F}, \tau) = \textsc{Cost}(W, cl^{\tau})$ and that for *any* clustering cl of W there exists a solution τ_{cl} to $\mathcal{F}(W)$ for which $cl = cl^{\tau_{cl}}$.

The transitive encoding of correlation clustering can be seen as a MaxSAT reformulation of a previously proposed integer programming model for correlation clustering, a model originally proposed for the clique partitioning problem [168, 175, 246]. The variables of $\textsc{Transitive}(W)$ are of form x_{ij} for each distinct pair $i, j = 1, \ldots, N$. Given a solution τ to $\textsc{Transitive}(W)$, the corresponding clustering cl^{τ} is constructed by assigning all v_i and v_j for which $\tau(x_{ij}) = 1$ to the same cluster. With these variables, the constraint $\textsc{IsFunc}(W)$ is encoded using $\theta(N^3)$ constraints of form $(x_{ij} \wedge x_{jk}) \rightarrow x_{ik}$ for distinct i, j, and k. Each such constraint corresponds to the clause $(\neg x_{ij} \vee \neg x_{jk} \vee x_{ik})$; all in all,

$$\textsc{IsFunc}(W) = \bigwedge_{i,j,k \text{ distinct}} (\neg x_{ij} \vee \neg x_{jk} \vee x_{ik}).$$

The two other constraints, $\textsc{SameCl}(i, j)$ and $\textsc{DiffCl}(i, j)$, are encoded with unit clauses: $\textsc{SameCl}(i, j) = (x_{ij})$ and $\textsc{DiffCl}(i, j) = (\neg x_{ij})$. In total $\textsc{Transitive}(W)$ contains $\theta(N^2)$ variables and $\theta(N^3)$ clauses.

The unary encoding of correlation clustering resembles to some extent a previously proposed quadratic integer programming formulation of correlation clustering [247]. In contrast to the transitive encoding, the unary encoding is parametrized on K, the maximum number of clusters in the solution clustering. The value of K needs to be set before creating the instance $\textsc{Unary}(W)$. In Paper V, we used $K = N$ in all experiments. This ensures that the produced clustering is an optimal solution to the general correlation clustering problem (recall the discussion at the end of Section 4.1.1). The main variables of $\textsc{Unary}(W)$ are the $\theta(N \cdot K)$ variables of form y_i^k for $1 \leq i \leq N$ and $1 \leq k \leq K$. Given a solution τ to $\textsc{Unary}(W)$, the clustering cl^{τ} is constructed using those variables by setting $cl^{\tau}(v_i) = k$ if and only if $\tau(y_i^k) = 1$. The constraint $\textsc{IsFunc}(W)$ is encoded using cardinality constraints:

$$\textsc{IsFunc}(W) = \bigwedge_{i=1}^{N} \text{CNF}(\sum_{k=1}^{K} y_i^k = 1).$$

In Paper V we used the so-called *sequential encoding* [215] to convert the cardinality constraints to CNF. The other two constraints, $\textsc{SameCl}(i, j)$ and $\textsc{DiffCl}(i, j)$, are encoded in a straightforward manner by

$$\text{SAMECL}(i, j) = \bigvee_{k=1}^{K} (y_i^k \wedge y_j^k) \text{ and } \text{DIFFCL}(i, j) = \bigwedge_{k=1}^{K} \neg(y_i^k \wedge y_j^k).$$

Including all clauses and variables introduced by the Tseitin encoding, the instance $\textsc{Unary}(W)$ contains $\theta(E \cdot K + N \cdot K)$ variables and $\theta(E \cdot K)$ clauses.

The third MaxSAT encoding we consider, the binary encoding, is essentially a bitwise reformulation of the unary encoding. Similarly to the unary encoding, the binary encoding is also parametrized on K. The main variables of $\textsc{Binary}(W)$ are $\theta(N \cdot \log_2(K))$ variables of form b_i^a for $1 \le i \le N$ and $1 \le a \le \lceil \log_2(K) \rceil$. Given a solution τ to $\textsc{Binary}(W)$, the clustering cl^τ is constructed by interpreting $\tau(b_i^{\lceil \log_2(K) \rceil}), \ldots, \tau(b_i^1)$ as a binary number c and setting $cl^\tau(v_i) = c$. We note that this construction results in a clustering $cl^\tau \colon V \to 2^{d'}$ for the smallest d' for which $2^{d'} \ge K$. In Paper V, we present extra constraints that can be added to $\textsc{Binary}(W)$ to ensure that $cl^\tau(v_i) \le K < N$ for all $1 \le i \le N$. Notice that such constraints are not needed if $K = N$, instead the cluster indexes of cl^τ can be permuted to the interval $0, \ldots, N-1$ as a post-processing step.

A convenient consequence of the construction of $\textsc{Binary}(W)$ is that $\textsc{IsFunc}(W) = \emptyset$, i.e., no clauses are required in order to ensure that cl^τ is a function. The encoding of $\textsc{SameCl}(i, j)$ and $\textsc{DiffCl}(i, j)$ is straightforward: two points v_i and v_j are assigned to the same cluster by cl^τ if and only if all bits in the binary representation of their cluster numbers are equal, i.e.,

$$\textsc{SameCl}(i, j) = \bigwedge_{k=1}^{\log_2(K)} (b_i^k \leftrightarrow b_j^k) \text{ and } \textsc{DiffCl}(i, j) = \bigvee_{k=1}^{\log_2(K)} \neg(b_i^k \leftrightarrow b_j^k).$$

In total $\textsc{Binary}(W)$ contains $\theta(E + N \cdot \log_2 K)$ variables and $\theta(E \cdot \log_2 K)$ clauses.

In Paper V we also consider different types of redundant constraints designed to reduce the symmetries in the binary encoding. As discussed in the previous section, the space of clusterings of W is very symmetric. Several of the symmetries are transferred to the space of MaxSAT solutions of $\textsc{Binary}(W)$. Some of the symmetries can be broken by adding extra constraints to $\textsc{Binary}(W)$. For a simple example of such an constraint, we can enforce the point v_1 to be assigned to cluster 0 with the constraint $\bigwedge_{k=1}^{\lceil \log_2(K) \rceil} (\neg b_1^k)$. Adding extra symmetry breaking constraints to the instance is non-trivial in general. While such constraints have the potential of decreasing the solving time of the instance, adding too many extra constraints can instead increase the size of the instance enough to degrade the performance of the MaxSAT solver. Notice for example that the transitive encoding naturally breaks all symmetries related to cluster indexing while being significantly larger than the other two instances. In Paper V we report on an experimental evaluation of some possible symmetry breaking constraints that could be used in conjunction with the binary encoding.

4.1.3 Experimental Evaluation

In Paper V we report on an experimental evaluation of the applicability of MaxSAT for solving correlation clustering. As benchmarks we used four sets of similarity values between amino-acid sequences of proteins [248], and seven different benchmark sets from the UCL machine learning repository [249]. The number of data points in the benchmark sets ranges from 327 to 990. In this section we overview the most significant results of the experiments using two of the four protein datasets, which we will from now on refer to as protein 1 and 2. The protein 1 dataset contains 669 data points and the protein 2 dataset contains 586 data points. The similarity values between the amino acid sequences in the sets were originally computed using BLAST [250], and the datasets were obtained from `http://www.paccanarolab.org/scps`.

In our experiments we compare our MaxSAT encodings with other previously proposed exact approaches to correlation clustering: an the integer linear programming (ILP) model [168, 175] and an quadratic integer programming (QIP) model [247]. All ILP and QIP models were solved with IBM CPLEX (version 12.6) and Gurobi Optimizer (version 6.0) using default settings. In our evaluation, all MaxSAT instances were solved using the 2013 evaluation version of MaxHS [121, 122]. The choice of MaxHS was motivated by it performing better than Eva500 [115], MsUnCore BCD2 version [110, 112], OpenWBO [113, 117] and ILP2013 [251] in preliminary experimentation. A per-instance time limit of 8 h was enforced on all solver runs. In addition to exact approaches we also compare our MaxSAT encodings with four polynomial-time inexact algorithms: KwickCluster (KC) [168], SDPC [252] and SCPS and CCA [248]. Out of these, the KC and SDPC algorithms were proposed for the general correlation clustering problem, while SCPS and CCA were proposed specifically to cluster the protein datasets. The semidefinite programs of SDPC were solved with the Matlab package SeDuMi 1.3 [253].

The first experiment we report on investigated the scalability of the exact approaches with respect to the number of data points in the input instance. For an increasing n, we formed a pruned similarity matrix W^n by taking the n first data points of the protein 1 dataset. Figure 4.2 shows the result of this test with n, the number of datapoints used, on the x-axis and the time required to solve W^n on the y-axis. The reason for the QIP model missing from the plot is that neither CPLEX nor Gurobi could solve any of the instances within 8 h, an observation we verified using the non-commercial SCIP [254] solver as well. From the figure we can clearly see that the binary MaxSAT encoding scales the best and is the only one able

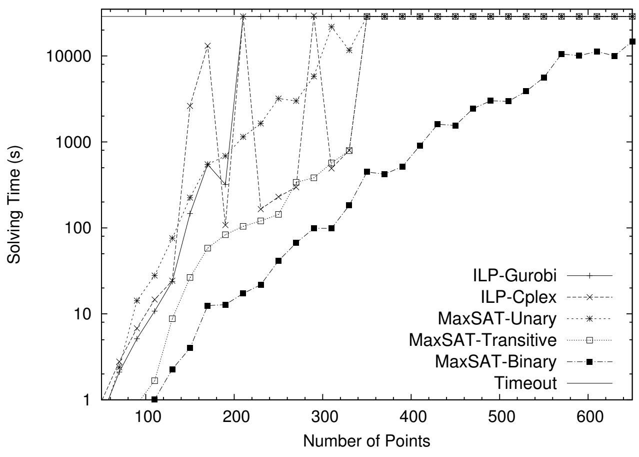

Figure 4.2: Point scalability of the exact approaches on the Protein 1 dataset (from Paper V).

to solve the full protein 1 dataset. Furthermore, many of the failed runs of the other exact approaches were due to memory-outs, suggesting that the algorithms would not terminate regardless of the time limit used.

The second experiment we report on was designed to investigate the scalability and quality of solutions obtained by the binary MaxSAT encoding with respect to the number of nonzero values in the similarity matrix. For $p \in \{0.05, 0.10, \ldots, 1\}$, we pruned the input similarity matrix W generated from either the protein 1 or protein 2 dataset by independently putting each nonzero value $W(i,j)$ to 0 with probability p. This approach results in a similarity matrix W' where the expected number of nonzero entries is $(1-p) \cdot 100\%$ of the number of nonzero entries in W. As can be seen from the top row of Figure 4.3, instances with fewer nonzero values are faster to solve with MaxHS. While this is hardly surprising, a more interesting observation of this experiment concerns the cost of the clusterings obtained by solving sparse instances. The bottom row of Figure 4.3 shows the cost $\textsc{Cost}(W, cl^p)$ of the optimal clustering cl^p of the pruned instance W' with respect to the complete instance W. The costs are plotted both for the exact MaxSAT method and the approximative algorithms KC, SDPC, SCPS and CCA. For both protein 1 and 2, we found that the clustering obtained by MaxSAT invoked on an instance with 60% ($p = 0.4$) of the non-zero values pruned was of lower cost than the clustering obtained by any of the inexact algorithms when invoked on the complete instance. These results suggest that first pruning a significant amount of the non-zero values of

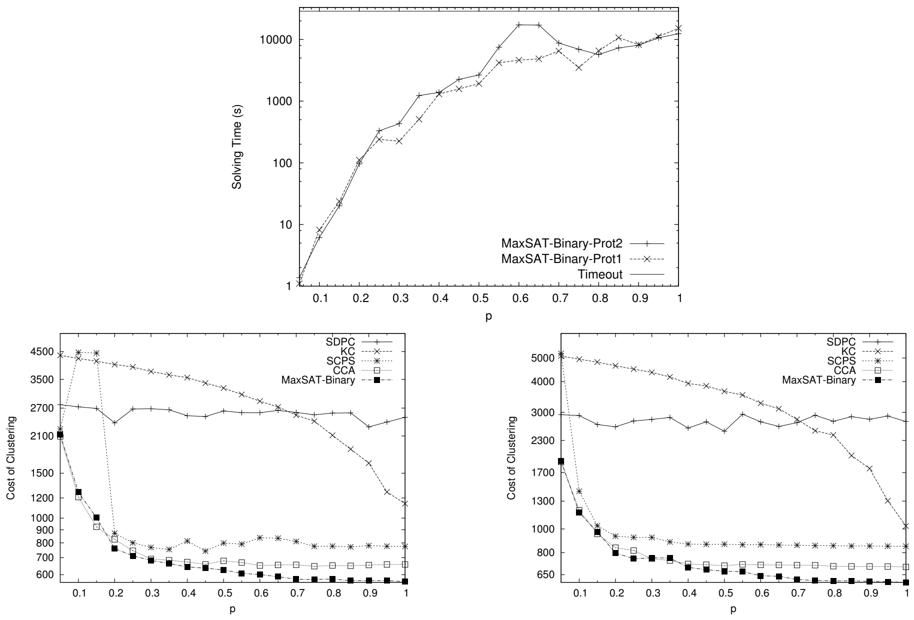

Figure 4.3: Top: Evolution of running times. Bottom: Cost of the clusterings obtained on sparse matrices. Bottom left: Protein 1 (669 datapoints), bottom right: Protein 2 (586 datapoints).

the matrix and then solving the sparser matrix using the binary MaxSAT encoding results in a competitive inexact approach to applications of correlation clustering with medium-size instances as long as running times in the order of a few minutes are acceptable. For applications where faster running times are required or larger instances need to be solved, the other inexact algorithms should be considered. All runs of KC, SDPC, SCPS and CCA reported on in Paper V were completed within a few seconds.

In addition the experiments on unconstrained correlation clustering, Paper V also includes results of experiments on the *constrained correlation clustering* problem. Constrained correlation clustering extends the correlation clustering problem by allowing extra hard constraints in the instance. We report on experiments evaluating the effect of extra symmetry breaking constraints as well as user specified must-link and cannot-link constraints which might come from an domain expert. We found that in many settings, adding extra constraints to the instance decreases the running time of the MaxSAT solver while not significantly increasing the cost of the produced clustering. As far as we are aware, adding similar constraints to the inexact algorithms is non-trivial. This further highlights the benefits of a declarative approach to solving correlation clustering.

4.2 Bounded Treewidth Bayesian Network Structure Learning

In Paper VI we propose and evaluate a MaxSAT encoding of the bounded treewidth Bayesian network structure learning problem (BTBNSL). We compare the resulting MaxSAT-based approach to the dynamic programming algorithm of [198], which at the time of publication of Paper VI was the only other known implementation of a solution algorithm to BTBNSL. Since the publication of Paper VI there has been a continued interest in BTBNSL [200, 255–257]. For example, an integer programming-based solution algorithm was published concurrently with Paper VI [199].

Given a set of observations (data) $\mathcal{D}$ over some set X of random variables, the goal of Bayesian network structure learning is to compute a Bayesian network structure which summarizes statistical dependencies and independencies in the data. BTBNSL further restricts the set of feasible solutions to networks that have treewidth less than some given bound $k \in \mathbb{N}$. In the score-based approach to BTBNSL, which we focus on, a scoring function $\textsc{Score}$ is precomputed based on the data. The scoring function assigns a score $\textsc{Score}(G)$ to each possible network structure $G = (X, E)$. The score measures how well G explains the data, an optimal network minimizes $\textsc{Score}$ over all possible networks. Our MaxSAT encoding is applicable under any *decomposable* scoring function. We give a precise definition of a scoring function being decomposable in the next section and note here that several commonly used scoring functions are decomposable, including MDL [258], BD [259], and fNML [11]. In the rest of the section, we assume that all scores are given as input and work with a generic decomposable scoring function $\textsc{Score}$.

This section is organized as follows. In Section 4.2.1 we detail BTBNSL and discuss how the treewidth of a Bayesian network can be computed. The MaxSAT encoding of BTBNSL is presented in Section 4.2.2 and an overview of the results of the experimental evaluation conducted in Paper VI is given in Section 4.2.3.

4.2.1 Problem Setting

Let $X = \{X_1, \ldots, X_N\}$ be a set of N random variables, and for each $i = 1, \ldots, N$, let $\mathcal{P}_i = 2^{X \setminus \{X_i\}}$ be the set of *candidate parent sets* of X_i.

An instance of BTBNSL consists of X, a bound $k \in \mathbb{N}$, and for each $i = 1, \ldots, N$ the set $\mathcal{P}_i$ as well as a *local score function* $s_i \colon \mathcal{P}_i \to \mathbb{N}$ associating a positive cost $s_i(P)$ to each $P \in \mathcal{P}_i$. Picking a single $P_i \in \mathcal{P}_i$ for each X_i gives rise to the directed graph $G = (X, E)$ in which $(X_j, X_i) \in E$ if and only if $X_j \in P_i$. We say that any X_j for which $(X_j, X_i) \in E$ is a parent of X_i, and X_i is a child of X_j. The graph G is a solution to the BTBNSL instance if it is acyclic and has treewidth less than k, i.e., if $\mathrm{TW}(G) \leq k$. The score $\textsc{Score}(G)$ of a solution G is equal to the sum of the local scores of each node and its parent set:

$$\textsc{Score}(G) = \sum_{X_i \in X} s_i(P_i). \tag{4.1}$$

As a side note, we say that any scoring function $\textsc{Score}$ for which the value $\textsc{Score}(G)$ can be computed similarly to Equation 4.1, is *decomposable.* A solution G^o is optimal if $\textsc{Score}(G^o) \leq \textsc{Score}(G)$ for any solution G.

Figure 4.4 illustrates the definition and computation of the treewidth of a Bayesian network $G = (X, E)$ [202, 260]. The treewidth of G is equal to the treewidth of its (undirected) moralized graph $\textsc{Moral}(G) = (X, E^M)$, obtained from G by adding an edge between any two nodes X_i and X_k that share a common child and dropping the direction of all edges. Given a linear ordering $\prec$ of X and two nodes $X_i, X_j \in X$, the node X_i is a *predecessor* of X_j (under $\prec$) if $X_i \prec X_j$ and $\{X_i, X_j\} \in E^M$. The undirected triangulation $\Delta(\textsc{Moral}(G), \prec)$ of $\textsc{Moral}(G)$ under $\prec$ is obtained by iteratively adding edges to E^M between pairs X_j and X_k of nodes that share a common predecessor. The edges are added until fix point. Finally, the directed ordered graph $\vec{\Delta}(\textsc{Moral}(G), \prec)$ is obtained from the resulting triangulation by ordering all edges according to $\prec$. The width of $\prec$ is the maximum out-degree of any node of $\vec{\Delta}(\textsc{Moral}(G), \prec)$. The treewidth $\mathrm{TW}(G)$ of G is the minimum-width of all linear orderings of X.

4.2.2 MaxSAT Encoding of BTBNSL

In this section we overview our MaxSAT encoding of BTBNSL. Given an instance of BTBNSL over $X = \{X_1, \ldots, X_N\}$ and a bound $k \in \mathbb{N}$, the encoding produces the MaxSAT instance $\textsc{Bayes}(X, k) = (F_h^X, F_s^X, w)$. For each variable $X_i \in X$ and potential parent set $S \in \mathcal{P}_i$, the instance $\textsc{Bayes}(X, k)$ includes a variable P_i^S. Given a solution τ to $\textsc{Bayes}(X, k)$, the graph $G^\tau = (X, E^\tau)$ corresponding to τ has $(X_j, X_i) \in E^\tau$ if and only if $X_j \in S$ for a $S \in \mathcal{P}_i$ for which $\tau(P_i^S) = 1$. The hard clauses of $\textsc{Bayes}(X, k)$ enforce that G^τ is a solution to X. The soft clauses ensure that $\textsc{Cost}(\mathcal{F}, \tau) = \textsc{Score}(G^\tau)$.

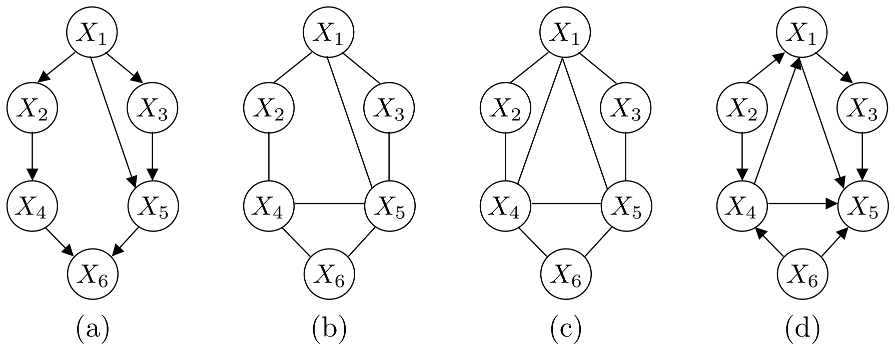

Figure 4.4: Computing the treewidth of a Bayesian network structure $G = (X = \{X_1, \ldots, X_6\}, E)$. (a) G; (b) the moralized graph $\textsc{Moral}(G) = (X, \mathrm{M}(E))$ of G; (c) the triangulation $\Delta(\textsc{Moral}(G), \prec)$ of the moralized graph under the linear ordering $X_6 \prec X_2 \prec X_4 \prec X_1 \prec X_3 \prec X_5$; (d) the ordered graph $\vec{\Delta}(\textsc{Moral}(G), \prec)$.

Next we overview the structure of $\textsc{Bayes}(X, k)$. Compared to the MaxSAT encodings for correlation clustering, the CNF conversions of the constraints in $\textsc{Bayes}(X, k)$ are somewhat involved. We refer the reader to Paper VI for the details. The hard clauses F_h^X of $\textsc{Bayes}(X, k)$ enforce that the graph G^τ corresponding to a solution τ of $\textsc{Bayes}(X, k)$ is a solution to the BTBNSL instance X. The clauses in F_h^X represent a conjunction of three different complex constraints:

$$F_h^X = \bigwedge_{i=1}^{N} \mathrm{CNF}(\sum_{S \in \mathcal{P}_i} P_i^S = 1) \wedge \mathrm{ACYC}(X) \wedge \mathrm{TWCNF}(X, k).$$

The constraint $\mathrm{CNF}(\sum_{S \in \mathcal{P}_i} P_i^S = 1)$ is satisfied if and only if G^τ has a single parent set S for each X_i. In Paper VI, we used the improved sequential counter [261] to encode this cardinality constraint to CNF.

The constraint $\mathrm{ACYC}(X)$ is satisfied if and only if the graph G^τ is acyclic. The CNF encoding of $\mathrm{ACYC}(X)$ assigns a level number $l(X_i) \in \mathbb{N}$ to each node $X_i \in X$ and enforces that the level number of a parent X_j of a node X_i satisfies $l(X_j) \leq l(X_i)$. In more detail, we model the level number $l(X_i)$ of each node X_i in binary using $\log_2(N)$ variables $b_i^1, \ldots, b_i^{\log_2(N)}$. We enforce $l(X_j) \leq l(X_i)$ by modeling the statement "the most significant bit in which the binary representations of $l(X_i)$ and $l(X_j)$ differ is 1 in $l(X_i)$".

The constraint $\mathrm{TWCNF}(X, k)$ is satisfied if and only if $\mathrm{TW}(G^\tau) \leq k$. The CNF translation of $\mathrm{TWCNF}(X, k)$ follows the SAT encoding for computing the treewidth of a fixed graph presented in [261]. Essentially, we enforce the existence of a linear ordering of the variables in X that has width at most k. This is enough to ensure that $\mathrm{TW}(G^{\tau}) \leq k$ as the treewidth is equal to the minimum width over all possible orderings.

The soft clauses F_s^X contain a unit negation $(\neg P_i^S)$ with weight $s_i(S)$ for all $i = 1, \ldots, N$ and $S \in \mathcal{P}_i$. These ensure that the cost incurred by selecting S as a parent for X_i is $s_i(S)$, as expected. These clauses ensure that $\textsc{Cost}(\mathcal{F}, \tau) = \textsc{Score}(G^{\tau})$ as required to make sure that the Bayesian network structure G^{τ} corresponding to an optimal solution τ to $\textsc{Bayes}(X, k)$ is an optimal solution to BTBNSL.

4.2.3 Experimental Evaluation

In Paper VI we report on an experimental evaluation evaluating of the applicability of MaxSAT for solving BTBNSL. As benchmark sets we used eight well-known UCl datasets [249] over 9–29 random variables, as well as two datasets (Adult and Housing) from [198]. Table 4.1 summarizes the benchmark sets. For each benchmark, we used the (decomposable) MDL scoring function [258]. We compare our MaxSAT encoding with the dynamic programming (DP) algorithm for BTBNSL of [198], the only other implementation of an exact algorithm for BTBNSL we were aware of at the time. All MaxSAT instances were solved with the MaxHS [121] solver. A per instance time limit of 8 h and memory limit of 30 GB were enforced on all benchmarks and solver runs. We used $k = 2, 3$ and 4 as bounds for the treewidth of the solution network.

Table 4.1 overviews the results of our experiment. For each treewidth bound, the best running time to find an optimal solution is highlighted in boldface. We observe that the dynamic programming approach is competitive with our MaxSAT approach only for the smallest dataset with 9 variables. Apart from the multiple timeouts ("> 28 800"), we observe that DP most often runs out of memory ("mo") on the datasets with more variables, especially for treewidth bounds greater than 2. In contrast, the MaxSAT approach (MS) timeouts on only two instances, and, especially, does not run out of memory. For a clear 2/3 majority of the instances, MS produces an optimal solution within half-an-hour; and for half of the instances within around 10 minutes.

Table 4.1: Running times in seconds of our MaxSAT-based approach (**MS**) and the dynamic programming (**DP**) approach [198] for different UCI datasets and treewidth bounds $k = 2, 3, 4$. Explanations: "mo" denotes a memory out; N denotes the number of variables (nodes); **#fails** denotes the number of times the memory or time limit was exceeded.

		treewidth ≤ 2		**treewidth** ≤ 3		**treewidth** ≤ 4		**#fails**	
Dataset	N	**MS** (s)	**DP** (s)	**MS** (s)	**DP** (s)	**MS** (s)	**DP** (s)	**MS**	**DP**
Abalone	9	64	**7**	166	**57**	**215**	536	0	0
Housing	14	**2 226**	6 927	**2 329**	$>$ 28 800	**2 991**	mo	**0**	2
Wine	14	**27**	6 924	**22**	$>$ 28 800	**171**	mo	**0**	2
Adult	15	**998**	$>$ 28 800	**1 623**	$>$ 28 800	**1 782**	mo	**0**	3
Voting	17	**22 909**	$>$ 28 800	**26 419**	mo	$>$ 28 800	mo	**1**	3
Zoo	17	**410**	$>$ 28 800	**412**	mo	**105**	mo	**0**	3
Hepatitis	20	**315**	mo	**100**	mo	**1 164**	mo	**0**	3
Heart	23	**1 198**	mo	**2 186**	mo	**41**	mo	**0**	3
Horse	28	**192**	mo	$>$ 28 800	mo	**544**	mo	**1**	3
Flag	29	**1 418**	mo	**11 148**	mo	**1 356**	mo	**0**	3
#fails:		**0**	7	**1**	9	**1**	9	**2**	25

Chapter 5

Conclusion

This thesis contributed to declarative methods for exactly solving combinatorial optimization problems. We focused on MaxSAT encodings and re-encodings of combinatorial optimization problems with the aim of solving instances that correspond to real-world applications.

In Papers I-IV we studied MaxSAT solving technology in the form of solver independent re-encodings, i.e., preprocessing, of MaxSAT instances $\mathcal{F}$ to other instances $pre(\mathcal{F})$ with the aim of making the time required to re-encode $\mathcal{F}$ and solve $pre(\mathcal{F})$ less than the time required to solve $\mathcal{F}$. In Papers I and II we further developed the previously proposed labeled CNF framework for MaxSAT preprocessing. In Paper I we showed that the extra label variables introduced during label-based preprocessing can be reused as assumption variables in many core-guided and implicit hitting set MaxSAT solvers, thus avoiding all variables that otherwise would be introduced by the solvers. We also showed that reusing labels as assumptions is necessary in order to improve the empirical performance of LMHS, an implicit hitting set based MaxSAT solver.

In Paper II we generalized the idea proposed in Paper I further by showing that some literals from the input MaxSAT instance itself can be used as labels during preprocessing and assumptions during solving. We demonstrated that such literals can be identified using simple pattern matching, resulting in a procedure we call group detection. Our empirical results indicate that group detection identifies a significant fraction of the literals in the evaluation benchmarks and that reusing detected literals results in modest further improvements to the empirical performance of LMHS.

Even though the ideas presented in Papers I and II are theoretically applicable to both core-guided and implicit hitting set based MaxSAT solvers, in practice we observed a more significant benefit of label-based preprocessing in conjunction with implicit hitting set based MaxSAT solvers. The relationship between label-based preprocessing and the formula rewriting performed by core-guided solvers remains an open and interesting question. A better understanding of the formula rewriting could result in improved performance of core-guided solvers and label-based preprocessing. Another approach to further improving the performance label-based preprocessing in SAT-based MaxSAT solvers could be via some form of *inprocessing*, i.e via preprocessing steps interleaved with the execution of the solving algorithm. It could also be interesting to investigate if similar ideas could be used in order to develop preprocessing in constraint programming [262, 263] or other constraint optimization paradigms.

In Paper III we presented the results of a theoretical analysis on the effect of label-based preprocessing with SAT-based preprocessing rules on core-guided and implicit hitting set based MaxSAT solvers. We showed that preprocessing can not decrease the number of iterations (SAT solver calls) required by either algorithm, but can help them avoid some long executions. As discussed at the end of Section 3.3, the results of Paper III highlight some potentially beneficial approaches to the further development of MaxSAT preprocessing techniques. Since preprocessing has limited effect on the number of iterations of MaxSAT solvers, the overall benefit of preprocessing on MaxSAT solving could be further improved by developing preprocessing techniques that allow faster core-extraction from unsatisfiable instances. Furthermore, even though label-based preprocessing with SAT-based preprocessing rules does not affect the MCS structure of MaxSAT instances, it can still affect the solutions corresponding to the MCSes. A better understanding of the effect that preprocessing has on the minimal solutions that correspond to optimal MCSes could result in improved preprocessing techniques.

It should also be mentioned that the base of the results presented in Paper III (Corollary 1 in Section 3.3) is actually stronger than what is required for sound MaxSAT preprocessing. An argument similar to the proof of Theorem 4 can be used to show that label-based preprocessing with SAT-based preprocessing rules preserves all minimal solutions to MaxSAT instances, not only the optimal ones. Thus preprocessing rules that are sound but do not satisfy Corollary 1 could affect the number of iterations of SAT-based MaxSAT solvers more significantly. Finally, it should be noted that the abstraction of core-guided solvers considered Paper III does not cover more recently proposed solvers, specifically the ones that introduce soft cardinality constraints. The effect of soft cardinality constraints on the MUS structure of MaxSAT instances remains an interesting open question. Developing a better understanding of how the formula rewritings used by core-guided solvers affect the MCSes of the instance could potentially improve both MaxSAT preprocessing and MaxSAT solving.

As the final contribution to MaxSAT preprocessing of this thesis, we proposed subsumed label elimination (SLE) in Paper IV. We showed that SLE is theoretically orthogonal to the SAT-based preprocessing rules in the sense that using SLE together with the SAT-based preprocessing rules can result in additional clauses and variables removed during preprocessing. We also demonstrated that, even though SLE is sound for MaxSAT, it does not preserve all MCSes of MaxSAT instances. Hence, an interesting further research direction would be to investigate the effect that SLE has on the number of iterations of SAT-based MaxSAT solvers. We hypothesize that the effect of SLE is similar to that of SAT-based preprocessing rules, but do not have a proof at this time. In addition to the theoretical results, Paper IV also demonstrated empirically that using SLE together with SAT-based preprocessing rules results in more variables and clauses being removed during preprocessing and in a decrease of the overall solving time of LMHS. These observations motivate further development of MaxSAT-specific preprocessing rules that make direct use of the label variables and weights of the soft clauses.

Papers V and VI proposed MaxSAT encodings of two data analysis tasks: correlation clustering and bounded treewidth Bayesian network structure learning. We empirically compared our MaxSAT-based solution approach with other, previously proposed exact algorithms. For both problems, we observed that the MaxSAT-based approach was faster and more memory efficient than the other considered approaches on several benchmarks. After the publication of Papers V and VI we have compiled a set of MaxSAT benchmarks of both correlation clustering and BTBNSL to each MaxSAT evaluation organized since 2015, i.e., the 2015, 2016 and 2017 evaluations. Interestingly, the solver that was most successful on those benchmarks in each evaluation was implicit hitting set based, an observation we made already in the original publications. This seems to suggest that these benchmarks exhibit some form of specific structure which is more easily exploited implicit hitting set based solvers compared to core-guided solvers. One possible explanation is the high diversity of weights of the soft clauses in the instances, which means that core-guided solvers need to perform a significant amount of clause cloning when solving them. A deeper understanding of the similarities and differences between core-guided and implicit hitting set based solvers remains an interesting open question for developing more effective encodings and MaxSAT solvers for combinatorial optimization problems at large.

References

[1] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., 1982.

[2] Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 19, pages 613–631. IOS Press, 2009.

[3] Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, 9(1):61–63, 1962.

[4] Nico L.J. Ulder, Emile H.L. Aarts, Hans-Jürgen Bandelt, Peter J.M. Van Laarhoven, and Erwin Pesch. Genetic local search algorithms for the traveling salesman problem. In *Proceedings of the 1st Workshop on the Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 109–116. Springer, 1990.

[5] Mauricio G.C. Resende and Panos M. Pardalos. *Handbook of Optimization in Telecommunications.* Springer Science & Business Media, 2008.

[6] Zhibin Wang, Chongzhi Zang, Jeffrey A. Rosenfeld, Dustin E. Schones, Artem Barski, Suresh Cuddapah, Kairong Cui, Tae-Young Roh, Weiqun Peng, Michael Q. Zhang, and Keji Zhao. Combinatorial patterns of histone acetylations and methylations in the human genome. *Nature Genetics*, 40:897–903, 2008.

[7] David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steven David Prestwich, Thomas Schiex, and Seydou Traoré. Computational protein design as an optimization problem. *Artificial Intelligence*, 212:59–79, 2014.

[8] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70–94, 2017.

[9] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based framework for efficient constrained clustering. In *Proceedings of the SIAM International Conference on Data Mining*, pages 94–105. SIAM, 2010.

[10] Sean Gilpin and Ian Davidson. A flexible ILP formulation for hierarchical clustering. *Artificial Intelligence*, 244:95–109, 2017.

[11] Tomi Silander, Teemu Roos, Petri Kontkanen, and Petri Myllymäki. Factorized normalized maximum likelihood criterion for learning Bayesian network structures. In *Proceedings of the 4th European Workshop on Probabilistic Graphical Models*, pages 257–272, 2008.

[12] Dag Sonntag, Matti Järvisalo, José M. Peña, and Antti Hyttinen. Learning optimal chain graphs with answer set programming. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, pages 822–831. AUAI Press, 2015.

[13] Antti Hyttinen, Paul Saikko, and Matti Järvisalo. A core-guided approach to learning optimal causal graphs. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 645–651. AAAI Press, 2017.

[14] Andreas Niskanen, Johannes Peter Wallner, and Matti Järvisalo. Optimal status enforcement in abstract argumentation. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 1216–1222. IJCAI/AAAI Press, 2016.

[15] Tias Guns, Anton Dries, Siegfried Nijssen, Guido Tack, and Luc De Raedt. MiningZinc: A declarative framework for constraint-based mining. *Artificial Intelligence*, 244:6–29, 2017.

[16] Tias Guns, Siegfried Nijssen, and Luc De Raedt. K-pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):402–418, 2013.

[17] Benjamin Négrevergne, Anton Dries, Tias Guns, and Siegfried Nijssen. Dominance programming for itemset mining. In *Proceedings of the 13th International Conference on Data Mining*, pages 557–566. IEEE Computer Society, 2013.

[18] John O. R. Aoga, Tias Guns, and Pierre Schaus. An efficient algorithm for mining frequent sequence with constraint programming. In *Machine Learning and Knowledge Discovery in Databases*, pages 315–330, Cham, 2016. Springer International Publishing.

[19] Thomas Hofmann and Joachim M. Buhmann. Pairwise data clustering by deterministic annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):1–14, 1997.

[20] Kerstin Bunte, Matti Järvisalo, Jeremias Berg, Petri Myllymäki, Jaakko Peltonen, and Samuel Kaski. Optimal neighborhood preserving visualization by maximum satisfiability. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1694–1700. AAAI Press, 2014.

[21] Sigurdur Olafsson, Xiaonan Li, and Shuning Wu. Operations research and data mining. *European Journal of Operational Research*, 187(3):1429–1448, 2008.

[22] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., 2004.

[23] Jussi Rintanen. Planning with SAT, admissible heuristics and A*. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 2015–2020. AAAI Press, 2011.

[24] Lei Zhang and Fahiem Bacchus. MaxSAT heuristics for cost optimal planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence.* AAAI Press, 2012.

[25] Luis C. Rabelo and Albert Jones. Job shop scheduling. In *Encyclopedia of Operations Research and Management Science*, pages 817–830. Springer, 2013.

[26] Mirko Stojadinovic. Air traffic controller shift scheduling by reduction to CSP, SAT and SAT-related problems. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming*, volume 8656 of *Lecture Notes in Computer Science*, pages 886–902. Springer, 2014.

[27] Miquel Bofill, Marc Garcia, Josep Suy, and Mateu Villaret. MaxSAT-based scheduling of B2B meetings. In *Proceedings of the 12th International Conference on the Integration of AI and OR Techniques in Constraint Programming*, volume 9075 of *Lecture Notes in Computer Science*, pages 65–73, 2015.

[28] Jan K. Lenstra, A.H.G. Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[29] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[30] Derya E. Akyol and G. Mirac Bayhan. A review on evolution of production scheduling with neural networks. *Computers & Industrial Engineering*, 53(1):95 – 122, 2007.

[31] Hui Xu, Rob A. Rutenbar, and Karem A. Sakallah. Sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):814–820, 2003.

[32] Samuel S. Brito, George H.G. Fonseca, Tulio A.M. Toffolo, Haroldo G. Santos, and Marcone J.F. Souza. A SA-VNS approach for the high school timetabling problem. *Electronic Notes in Discrete Mathematics*, 39:169–176, 2012.

[33] Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.

[34] Edmund K. Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

[35] Alain Hertz. Tabu search for large scale timetabling problems. *European Journal of Operational Research*, 54(1):39–47, 1991.

[36] Roberto J.A. Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, 218(1):71–91, 2014.

[37] Manu Jose and Rupak Majumdar. Cause clue clauses: Error localization using maximum satisfiability. *ACM SIGPLAN Notices*, 46(6):437–446, 2011.

[38] Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In *Proceedings of the 11th International Conference on Formal Methods in Computer-Aided Design*, pages 63–66. FMCAD Inc, 2011.

[39] Morten Mossige, Arnaud Gotlieb, and Hein Meling. Deploying constraint programming for testing ABB's painting robots. *AI Magazine*, 38(2):94–96, 2017.

[40] Javier Barbas and Angel Marin. Maximal covering code multiplexing access telecommunication networks. *European Journal of Operational Research*, 159(1):219 – 238, 2004.

[41] Dimitris Bertsimas, Guglielmo Lulli, and Amedeo R. Odoni. An integer optimization approach to large-scale air traffic flow management. *Operations Research*, 59(1):211–227, 2011.

[42] Arthur Richards and Jonathan P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference*, volume 3, pages 1936–1941. IEEE, 2002.

[43] Pey-Chang Lin and Sunil Khatri. Application of MaxSAT-based ATPG to optimal cancer therapy design. *BMC Genomics*, 13(6), 2012.

[44] Tung-Wei Kuo, Kate Ching-Ju Lin, and Ming-Jer Tsai. On the construction of data aggregation tree with minimum energy cost in wireless sensor networks: NP-completeness and approximation algorithms. *IEEE Transactions on Computers*, 65(10):3109–3121, 2016.

[45] Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for optimal decision trees and adaptive TSP problems. *Mathematics of Operations Research*, 42(3):876–896, 2017.

[46] Anton Milan, Seyed Hamid Rezatofighi, Ravi Garg, Anthony R. Dick, and Ian D. Reid. Data-driven approximations to NP-hard problems. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 1453–1459. AAAI Press, 2017.

[47] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 38–49. ACM, 1973.

[48] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., 2001.

[49] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[50] Jordan Thayer and Wheeler Ruml. Anytime heuristic search: Frameworks and algorithms. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search*, pages 121–128. AAAI Press, 2010.

[51] Hisao Ishibuchi and Takashi Yamamoto. Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining. *Fuzzy sets and Systems*, 141(1):59–88, 2004.

[52] Eric A. Hansen and Rong Zhou. Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28:267–297, 2007.

[53] Gerhard J. Woeginger. *Exact Algorithms for NP-Hard Problems: A Survey*, pages 185–207. Springer Berlin Heidelberg, 2003.

[54] Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573, 2004.

[55] Fedor V. Fomin and Petteri Kaski. Exact exponential algorithms. *Communications of the ACM*, 56(3):80–88, 2013.

[56] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[57] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization.* Wiley Interscience Series in Discrete Mathematics and Optimization. Wiley, 1988.

[58] T.C. Hu and Andrew B. Kahng. Linear and integer programming in practice. In *Linear and Integer Programming Made Easy*, pages 117–130. Springer, 2016.

[59] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of Constraint Programming.* Elsevier, 2006.

[60] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[61] Piero Bonatti, Francesco Calimeri, Nicola Leone, and Francesco Ricca. Answer set programming. In *A 25-Year Perspective on Logic Programming*, pages 159–182. Springer-Verlag, 2010.

[62] Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint programming for data mining and machine learning. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. AAAI Press, 2010.

[63] Ronald de Haan, Martin Kronegger, and Andreas Pfandler. Fixed-parameter tractable reductions to SAT for planning. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 2897–2903. AAAI Press, 2015.

[64] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Hel. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[65] Mikoláš Janota, William Klieber, João Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. *Artificial Intelligence*, 234:1–25, 2016.

[66] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[67] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[68] James F. Campbell. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, 72(2):387–405, 1994.

[69] Michela Milano and Francesca Rossi. Constraint programming. *Intelligenza Artificiale*, 3(1-2):28–34, 2006.

[70] Daniel Larraz, Kaustubh Nimkar, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. Proving non-termination using MaxSMT. In *Proceedings of the 26th International Conference on Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 779–796, 2014.

[71] Roberto Sebastiani and Patrick Trentin. On optimization modulo theories, MaxSMT and sorting networks. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 10206 of *Lecture Notes in Computer Science*, pages 231–248. Springer, 2017.

[72] Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A tool for optimization modulo theories. In *Proceedings of the 27th International Conference on Computer Aided Verification*, volume 9206 of *Lecture Notes in Computer Science*, pages 447–454. Springer, 2015.

[73] Tomas Balyo, Marijn Heule, and Matti Järvisalo. SAT Competition 2016: Recent developments. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 5061–5063. AAAI Press, 2017.

[74] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. *AI Magazine*, 33(1):89–92, 2012.

[75] Jeremias Berg and Matti Järvisalo. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artificial Intelligence*, 244:110–142, 2017.

[76] Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 86–95. JMLR, 2014.

[77] Antti Hyttinen, Patrik O. Hoyer, Frederick Eberhardt, and Matti Järvisalo. Discovering cyclic causal models with latent variables: A general SAT-based procedure. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence.* AUAI Press, 2013.

[78] James D. Park. Using weighted MaxSAT engines to solve MPE. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 682–687. AAAI Press / The MIT Press, 2002.

[79] Tian Sang, Paul Beame, and Henry A. Kautz. A dynamic approach for MPE and weighted MaxSAT. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 173–179, 2007.

[80] Ana Graça, Inês Lynce, João Marques-Silva, and Arlindo L. Oliveira. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In *Revised Selected Papers of the 4th International Conference on Algebraic and Numeric Biology*, volume 6479 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2012.

[81] Ana Graça, João Marques-Silva, Inês Lynce, and Arlindo L. Oliveira. Haplotype inference with pseudo-Boolean optimization. *Annals of Operations Research*, 184(1):137–162, 2011.

[82] Inês Lynce and João Marques-Silva. Haplotype inference with Boolean satisfiability. *International Journal on Artificial Intelligence Tools*, 17(2):355–387, 2008.

[83] Xiaojuan Liao, Miyuki Koshimura, Hiroshi Fujita, and Ryuzo Hasegawa. MaxSAT encoding for MC-net-based coalition structure generation problem with externalities. *IEICE Transactions*, 97-D(7):1781–1789, 2014.

[84] Jeremias Berg and Matti Järvisalo. SAT-based approaches to treewidth computation: An evaluation. In *Proceedings of of the 26th International Conference on Tools with Artificial Intelligence*, pages 328–335. IEEE Computer Society, 2014.

[85] João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, volume 7514 of *Lecture Notes in Computer Science*, pages 941–956. Springer, 2012.

[86] Dawn M. Strickland, Earl R. Barnes, and Joel S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53(3):389–402, 2005.

[87] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 542–547. Morgan Kaufmann, 1999.

[88] João Marques-Silva, Mikolas Janota, Alexey Ignatiev, and Antonio Morgado. Efficient model based diagnosis with maximum satisfiability. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 1966–1972. AAAI Press, 2015.

[89] Alessandro Bezerra Trindade, Renato De Faria Degelo, Edilson Galvão Dos Santos Junior, Hussama Ibrahim Ismail, Helder Cruz Da Silva, and Lucas Carvalho Cordeiro. Multi-core model checking and maximum satisfiability applied to hardware-software partitioning. *International Journal of Embedded Systems*, 9(6):570–582, 2017.

[90] Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. Automated synthesis of semantic malware signatures using maximum satisfiability. In *Proceedings of the 24th Annual Network and Distributed System Security Symposium*. The Internet Society, 2017.

[91] Sean Safarpour, Hratch Mangassarian, Andreas G. Veneris, Mark H. Liffiton, and Karem A. Sakallah. Improved design debugging using maximum satisfiability. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design*, pages 13–19. IEEE Computer Society, 2007.

[92] Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik. Maximum satisfiability in software analysis: Applications and techniques. In *Proceedings of the 29th International Conference on Computer Aided Verification*, volume 10426 of *Lecture Notes in Computer Science*, pages 68–94. Springer, 2017.

[93] Christian Muise, J. Christopher Beck, and Sheila A. McIlraith. Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research*, 57:113–149, 2016.

[94] Marcel Kevin Tiepelt and Tilak Raj Singh. Finding pre-production vehicle configurations using a MaxSAT framework. In *Proceedings of the 18th International Configuration Workshop*, page 117. École des Mines d'Albi-Carmaux, 2016.

[95] Carlos Ansótegui, Idelfonso Izquierdo, Felip Manyà, and José Torres-Jiménez. A MaxSAT-based approach to constructing optimal covering arrays. In *Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence*, volume 256 of *Frontiers in Artificial Intelligence and Applications*, pages 51–59. IOS Press, 2013.

[96] Josep Argelich, Daniel Le Berre, Inês Lynce, João Marques-Silva, and Pascal Rapicault. Solving Linux upgradeability problems using Boolean optimization. In *Proceedings of the 1st International Workshop on Logics for Component Configuration*, volume 29 of *Electronic Proceedings in Theoretical Computer Science*, pages 11–22. Open Publishing Association, 2010.

[97] Yibin Chen, Sean Safarpour, João Marques-Silva, and Andreas G. Veneris. Automated design debugging with maximum satisfiability.

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29(11):1804–1817, 2010.

[98] Inês Lynce and João Marques-Silva. Restoring CSP satisfiability with MaxSAT. *Fundamenta Informaticae*, 107(2-3):249–266, 2011.

[99] Xiaojuan Liao, Hui Zhang, and Miyuki Koshimura. Reconstructing AES key schedule images with SAT and MaxSAT. *IEICE Transactions on Information and Systems*, 99(1):141–150, 2016.

[100] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second MaxSAT evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):251–278, 2008.

[101] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. MaxSAT Evaluations. `http://maxsat.ia.udl.cat/`.

[102] Carlos Ansótegui, Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2017, 2017. `http://mse17.cs.helsinki.fi/`.

[103] Antonio Morgado, Federico Heras, and João Marques Silva. Model-guided approaches for MaxSAT solving. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence*, pages 931–938. IEEE Computer Society, 2013.

[104] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A partial MaxSAT solver. *Journal of Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012.

[105] Daniel Le Berre and Anne Parrain. The SAT4J library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[106] Carlos Ansótegui and Joel Gabàs. WPM3: An (in) complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017.

[107] João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proceedings of Design, Automation and Test in Europe*, pages 408–413. ACM, 2008.

[108] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Improving linear search algorithms with model-based approaches for MaxSAT solving. *Journal of Experimental & Theoretical Artificial Intelligence*, 27(5):673–701, 2015.

[109] Vasco M. Manquinho, João Marques-Silva, and Jordi Planes. Algorithms for weighted Boolean optimization. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

[110] Federico Heras, Antonio Morgado, and João Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*. AAAI Press, 2011.

[111] António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

[112] António Morgado, Federico Heras, and João Marques-Silva. Improvements to core-guided binary search for MaxSAT. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, volume 7317 of *Lecture Notes in Computer Science*, pages 284–297. Springer, 2012.

[113] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSAT. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming*, volume 8656 of *Lecture Notes in Computer Science*, pages 531–548. Springer, 2014.

[114] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[115] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2717–2723. AAAI Press, 2014.

[116] Zhaohui Fu and Sharad Malik. On solving the partial MaxSAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

[117] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014.

[118] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77 – 105, 2013.

[119] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A MaxSAT algorithm using cardinality constraints of bounded size. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 2677–2683. AAAI Press, 2015.

[120] Nikolaj Bjørner and Nina Narodytska. Maximum satisfiability using cores and correction sets. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 246–252. AAAI Press, 2015.

[121] Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSAT. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

[122] Jessica Davies and Fahiem Bacchus. Solving MaxSAT by solving a sequence of simpler SAT instances. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

[123] Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: a SAT-IP hybrid MaxSAT solver. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016.

[124] Brian Borchers and Judith Furman. A two-phase exact algorithm for MaxSAT and weighted MaxSAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 1998.

[125] Chu Min Li, Felip Manya, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound MaxSAT solvers. In *Proceedings of the 11th International Conference on Principles and*

Practice of Constraint Programming, volume 3709 of *Lecture Notes in Computer Science*, pages 403–414. Springer, 2005.

[126] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, volume 10, pages 128–133. AAAI Press, 2010.

[127] André Abramé and Djamal Habet. AHMAXSAT: Description and evaluation of a branch and bound MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:89–128, 2015.

[128] Yan-Li Liu, Chu-Min Li, Kun He, and Yi Fan. Breaking cycle structure to improve lower bound for MaxSAT. In *Proceedings of the 10th International Workshop on Frontiers in Algorithmics*, volume 9711 of *Lecture Notes in Computer Science*, pages 111–124. Springer, 2016.

[129] André Abramé and Djamal Habet. Learning nobetter clauses in MaxSAT branch and bound solvers. In *Proceedings of the 28th International Conference on Tools with Artificial Intelligence*, IEEE Computer Society, pages 452–459, 2016.

[130] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[131] Chu Min Li, Felip Manya, and Jordi Planes. New inference rules for MaxSAT. *Journal of Artificial Intelligence Research*, 30(1):321–359, 2007.

[132] Gintaras Palubeckis. A new bounding procedure and an improved exact algorithm for the Max-2-SAT problem. *Applied Mathematics and Computation*, 215(3):1106–1117, 2009.

[133] Zhao Xing and Weixiong Zhang. MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial intelligence*, 164(1-2):47–80, 2005.

[134] Han Lin, Kaile Su, and Chu Min Li. Within-problem learning for efficient lower bound computation in MaxSAT solving. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 351–356. AAAI Press, 2008.

[135] Anton Belov, António Morgado, and João Marques-Silva. SAT-based preprocessing for MaxSAT. In *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 8312 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2013.

[136] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[137] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers Inc., 1988.

[138] Gal Elidan and Stephen Gould. Learning bounded treewidth Bayesian networks. *Journal of Machine Learning Research*, 9:2699–2731, 2008.

[139] Martin W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS Journal on Computing*, 6(4):445–454, 1994.

[140] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.

[141] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[142] Marijn Heule, Matti Järvisalo, and Armin Biere. Covered clause elimination. In *Short papers for the 17th International Conference on Logic for Programming Artificial Intelligence, and Reasoning*, volume 13 of *EPiC Series in Computing*, pages 41–46. EasyChair, 2013.

[143] Cédric Piette, Youssef Hamadi, and Lakhdar Saïs. Vivifying propositional clausal formulae. In *Proceedings of the 18th European Conference on Artificial Intelligence*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 525–529. IOS Press, 2008.

[144] Matti Järvisalo and Armin Biere. Reconstructing solutions after blocked clause elimination. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing*, volume 6175 of *Lecture Notes in Computer Science*, pages 340–345. Springer, 2010.

[145] Inês Lynce and João Marques-Silva. Probing-based preprocessing techniques for propositional satisfiability. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 105–110. IEEE Computer Society, 2003.

[146] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Selected Revised Papers of the 6th International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 2004.

[147] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In *Online Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, pages 276–291. Springer, 2004.

[148] Roman Gershman and Ofer Strichman. Cost-effective hyper-resolution for preprocessing CNF formulas. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 423–429. Springer, 2005.

[149] Hyojung Han and Fabio Somenzi. Alembic: An efficient algorithm for CNF preprocessing. In *Proceedings of the 44th annual Design Automation Conference*, pages 582–587. ACM, 2007.

[150] Marijn Heule, Matti Järvisalo, and Armin Biere. Efficient CNF simplification based on binary implication graphs. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing*, volume 6695 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2011.

[151] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for MaxSAT. *Artificial Intelligence*, 171(8-9):606–618, 2007.

[152] Javier Larrosa, Federico Heras, and Simon de Givry. A logical approach to efficient MaxSAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

[153] Josep Argelich, Chu Min Li, and Felip Manyà. A preprocessor for MaxSAT solvers. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing*, volume 4996 of *Lecture Notes in Computer Science*, pages 15–20. Springer, 2008.

[154] Anton Belov, Matti Järvisalo, and João Marques-Silva. Formula preprocessing in MUS extraction. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 2013.

[155] Byron Ellis and Wing Hung Wong. Learning causal bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103(482):778–789, 2008.

[156] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. The global K-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[157] Leonard J. Schulman. Clustering for edge-cost minimization. *Electronic Colloquium on Computational Complexity (ECCC)*, 6(35), 1999.

[158] Anil K. Jain, M. Narasimha Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[159] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, Inc., 1988.

[160] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[161] Robert C. Edgar. Search and clustering orders of magnitude faster than blast. *Bioinformatics*, 26(19):2460–2461, 2010.

[162] Daniel Aloise, Pierre Hansen, and Leo Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, 131(1):195–220, 2012.

[163] Weifeng Zhi, Buyue Qian, and Ian Davidson. Scalable constrained spectral clustering via the randomized projected power method. In *Proceedings of the 2017 IEEE International Conference on Data Mining*, pages 1201–1206. IEEE Computer Society, 2017.

[164] Kamal Jain and Vijay V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 2–13. IEEE Computer Society, 1999.

[165] Rajkumar Jain and Narendra S. Chaudhari. Formulation of 3-clustering as a 3-SAT problem. In *Proceedings of the 5th Indian International Conference on Artificial Intelligence*, pages 465–472. IICAI, 2011.

[166] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. Constrained clustering using SAT. In *Proceedings of the 11th International Conference on Advances in Intelligent Data Analysis*, volume 7619 of *Lecture Notes in Computer Science*, pages 207–218. Springer, 2012.

[167] Marianne Mueller and Stefan Kramer. Integer linear programming models for constrained clustering. In *Proceedings of the 13th International Conference on Discovery Science*, volume 6332 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2010.

[168] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and Clustering. *Journal of the ACM*, 55(5):23:1–23:27, 2008.

[169] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.

[170] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[171] Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. *Theory of Computing*, 2(1):249–266, 2006.

[172] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.

[173] Nir Ailon and Edo Liberty. Correlation clustering revisited: The ”true” cost of error minimization problems. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 24–36. Springer, 2009.

[174] Erik D. Demaine and Nicole Immorlica. Correlation clustering with partial information. In *Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization*

Problems and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, volume 2764 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

[175] Jurgen Van Gael and Xiaojin Zhu. Correlation clustering for crosslingual link detection. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1744–1749. AAAI Press, 2007.

[176] Adnan Darwiche. Chapter 11 Bayesian networks. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 467 – 509. Elsevier, 2008.

[177] Sebastian Ordyniak and Stefan Szeider. Parameterized complexity results for exact Bayesian network structure learning. *Journal of Artificial Intelligence Research*, 46:263–302, 2013.

[178] Cassio P. de Campos and Qiang Ji. Efficient learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

[179] Luis M. de Campos. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research*, 7:2149–2187, 2006.

[180] Mark Bartlett and James Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

[181] David M. Chickering. Learning Bayesian networks is NP-complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

[182] Nir Friedman and Daphne Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50:95–125, 2003.

[183] David M. Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, 2002.

[184] Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proceedings of the 13th International Conference on Artificial Intelli-*

gence and Statistics, volume 9 of *JMLR Proceedings*, pages 358–365. JMLR, 2010.

[185] James Cussens. Bayesian network learning by compiling to weighted MaxSAT. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 105–112. AUAI Press, 2008.

[186] Alexandra M. Carvalho, Teemu Roos, Arlindo L. Oliveira, and Petri Myllymäki. Discriminative learning of Bayesian networks via factorized conditional log-likelihood. *Journal of Machine Learning Research*, 12:2181–2210, July 2011.

[187] Daniel Eaton and Kevin Murphy. Bayesian structure learning using dynamic programming and MCMC. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 101–108. AUAI Press, 2007.

[188] Sascha Ott and Satoru Miyano. Finding optimal gene networks using biological constraints. *Genome Informatics*, 14:124–133, 2003.

[189] Tomi Silander and Petri Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, pages 445–452. AUAI Press, 2006.

[190] James Cussens. Bayesian network learning with cutting planes. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 153–160. AUAI Press, 2011.

[191] Changhe Yuan and Brandon Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48:23–65, 2013.

[192] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393 – 405, 1990.

[193] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. In Glenn Shafer and Judea Pearl, editors, *Readings in Uncertain Reasoning*, pages 415–448. Morgan Kaufmann Publishers Inc., 1990.

[194] Johan Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In *Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 237–242. IOS Press, 2010.

[195] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[196] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1 – 45, 1998.

[197] Umberto Bertele and Francesco Brioschi. *Nonserial Dynamic Programming.* Academic Press, Inc., Orlando, FL, USA, 1972.

[198] Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics*, volume 31 of *JMLR Workshop and Conference Proceedings*, pages 370—378. JMLR, 2013.

[199] Pekka Parviainen, Hossein Shahrabi Farahani, and Jens Lagergren. Learning bounded tree-width Bayesian networks using integer linear programming. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 751–759. JMLR, 2014.

[200] Siqi Nie, Denis D. Mauá, Cassio P. De Campos, and Qiang Ji. Advances in learning Bayesian networks of bounded treewidth. In *Advances in Neural Information Processing Systems*, pages 2285–2293, 2014.

[201] Mukund Narasimhan and Jeff Bilmes. PAC-learning bounded tree-width graphical models. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 410–417. AUAI Press, 2004.

[202] Hans L. Bodlaender. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[203] Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[204] João Marques-Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227. IEEE Computer Society, 1996.

[205] Frank Hutter, Marius Lindauer, Adrian Balint, Sam Bayless, Holger Hoos, and Kevin Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artificial Intelligence*, 243:1–25, 2017.

[206] Forrest Sheng Bao, Chris Gutierrez, Jeriah Jn Charles-Blount, Yaowei Yan, and Yuanlin Zhang. Accelerating Boolean satisfiability (SAT) solving by common subclause elimination. *Artificial Intelligence Review*, pages 1–15, 2017.

[207] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[208] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, pages 399–404. Morgan Kaufmann Publishers Inc., 2009.

[209] Armin Biere. Lingeling, Plingeling and Treengeling entering the SAT competition 2013. In *Proceedings of SAT Competition*, volume B-2013-1 of *Department of Computer Science Series of Publications B*, pages 51–52. University of Helsinki, 2013.

[210] Gilles Audemard and Laurent Simon. Glucose in the SAT 2014 competition. *SAT COMPETITION 2014*, page 31, 2014.

[211] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient conflict-driven learning in a Boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*, pages 279–285. IEEE Computer Society, 2001.

[212] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 4, pages 131–153. IOS Press, 2009.

[213] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[214] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[215] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005.

[216] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[217] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer Berlin Heidelberg, 2003.

[218] Soukaina Hattad, Saïd Jabbour, Lakhdar Sais, and Yakoub Salhi. Enhancing pigeon-hole based encoding of Boolean cardinality constraints. In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, volume 2, pages 299–307. SciTePress, 2017.

[219] Toru Ogawa, Yangyang Liu, Ryuzo Hasegawa, Miyuki Koshimura, and Hiroshi Fujita. Modulo based CNF encoding of cardinality constraints and its application to MaxSAT solvers. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence*, pages 9–17. IEEE Computer Society, 2013.

[220] Ignasi Abío, Valentin Mayer-Eichberger, and Peter J. Stuckey. Encoding linear constraints with implication chains to CNF. In *Proceedings of the 21st International Conference on the Principles and Practice of Constraint Programming*, volume 9255 of *Lecture Notes in Computer Science*, pages 3–11. Springer, 2015.

[221] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2009.

[222] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial MaxSAT. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*. AAAI Press, 2010.

[223] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Technical Communications of the 28th International Conference on Logic Programming*, LIPIcs, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[224] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving SAT-based weighted MaxSAT solvers. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, volume 7514 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012.

[225] Jeremias Berg and Matti Järvisalo. Weight-aware core extraction in SAT-based MaxSAT solving. In *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming*, volume 10416 of *Lecture Notes in Computer Science*, pages 652–670. Springer, 2017.

[226] Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MaxSAT solving. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.

[227] Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in MaxSAT. In *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017.

[228] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient weighted MaxSAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[229] Rolf Niedermeier and Peter Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36(1):63–88, 2000.

[230] Anton Belov and João Marques-Silva. Generalizing redundancy in propositional logic: Foundations and hitting sets duality. *CoRR*, abs/1207.1257, 2012.

[231] Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2010.

[232] Federico Heras, Antonio Morgado, and João Marques Silva. MaxSAT-based encodings for Group MaxSAT. *AI Communications*, 28(2):195–214, 2015.

[233] Fahiem Bacchus and Nina Narodytska. Cores in core based MaxSAT algorithms: An analysis. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing*, volume 8561 of *Lecture Notes in Computer Science*, pages 7–15. Springer, 2014.

[234] Jessica Davies. *Solving MaxSAT by Decoupling Optimization and Satisfaction*. PhD thesis, University of Toronto, 2013.

[235] Alexey Ignatiev, Antonio Morgado, Vasco Manquinho, Ines Lynce, and João Marques-Silva. Progression in maximum satisfiability. In *Proceedings of the 21st European Conference on Artificial Intelligence*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 453–458. IOS Press, 2014.

[236] Olivier Coudert and Jean Christophe Madre. New ideas for solving covering problems. In *Proceedings of the 32st Conference on Design Automation*, pages 641–646. ACM Press, 1995.

[237] Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.

[238] Grigorii S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer Berlin Heidelberg, 1983.

[239] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, page 238. IEEE Computer Society, 2002.

[240] Divya Pandove, Rinkle Rani, and Shivani Goel. Local graph based correlation clustering. *Knowledge-Based Systems*, 138:155–175, 2017.

[241] Yixin Zhuang, Hang Dou, Nathan Carr, and Tao Ju. Feature-aligned segmentation using correlation clustering. *Computational Visual Media*, 3(2):147–160, 2017.

[242] Nate Veldt, Anthony Ian Wirth, and David F. Gleich. Correlation clustering with low-rank matrices. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1025–1034. ACM, 2017.

[243] Evgeny Levinkov, Alexander Kirillov, and Bjoern Andres. A comparative study of local search algorithms for correlation clustering. In *Proceedings of the 39th German Conference on Pattern Recognition*, volume 10496 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2017.

[244] Atsushi Miyauchi and Tomohiro Sonobeand Noriyoshi Sukegawa. Exact clustering via integer programming and maximum satisfiability. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages ??? – ??? AAAI Press, 2018. To appear.

[245] Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. In *Proceedings of the 17th International Conference on Artificial Intelligence*, pages 1103–1110. AAAI Press / The MIT Press, 2000.

[246] Martin Grötschel and Yoshiko Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1):59–96, 1989.

[247] Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Tao Jiang. On the approximation of correlation clustering and consensus clustering. *Journal of Computer and System Sciences*, 74(5):671–696, 2008.

[248] Tamás Nepusz, Rajkumar Sasidharan, and Alberto Paccanaro. SCPS: a fast implementation of a spectral method for detecting protein families on a genome-wide scale. *BMC Bioinformatics*, 11:120, 2010.

[249] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[250] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[251] Carlos Ansótegui and Joel Gabàs. Solving (weighted) partial MaxSAT with ILP. In *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 403–409. Springer, 2013.

[252] Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending grothendieck's inequality. In *Proceedings of the 45th Symposium on Foundations of Computer Science*, pages 54–60. IEEE Computer Society, 2004.

[253] Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Version 1.05 available from `http://fewcal.kub.nl/sturm`.

[254] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: A new approach to integrate CP and MIP. In *Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5015 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2008.

[255] Siqi Nie, Cassio P. De Campos, and Qiang Ji. Learning bounded treewidth Bayesian networks via sampling. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 387–396. Springer, 2015.

[256] Mauro Scanagatta, Giorgio Corani, Cassio P. de Campos, and Marco Zaffalon. Learning treewidth-bounded Bayesian networks with thousands of variables. In *Advances in Neural Information Processing Systems*, pages 1462–1470, 2016.

[257] Siqi Nie, Cassio P. de Campos, and Qiang Ji. Efficient learning of Bayesian networks with bounded tree-width. *International Journal of Approximate Reasoning*, 80:412–427, 2017.

[258] Wai Lam and Fahiem Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

[259] Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[260] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[261] Marko Samer and Helmut Veith. Encoding treewidth into SAT. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *Lecture Notes in Computer Science*, pages 45–50. Springer, 2009.

[262] Peter J. Stuckey. Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 5–9. Springer, 2010.

[263] Broes de Cat, Marc Denecker, Maurice Bruynooghe, and Peter J. Stuckey. Lazy model expansion: Interleaving grounding with search. *Journal of Artificial Intelligence Research*, 52:235–286, 2015.

Paper I

Jeremias Berg, Paul Saikko, and Matti Järvisalo

Improving the Effectiveness of SAT-Based Preprocessing for MaxSAT

Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Improving the Effectiveness of SAT-Based Preprocessing for MaxSAT

Jeremias Berg and **Paul Saikko** and **Matti Järvisalo**
HIIT & Department of Computer Science, University of Helsinki, Finland

Abstract

Solvers for the Maximum satisfiability (MaxSAT) problem find an increasing number of applications today. We focus on improving MaxHS—one of the most successful recent MaxSAT algorithms—via SAT-based preprocessing. We show that employing SAT-based preprocessing via the so-called labelled CNF (LCNF) framework before calling MaxHS can in some cases greatly degrade the performance of the solver. As a remedy, we propose a lifting of MaxHS that works directly on LCNFs, allowing for a tighter integration of SAT-based preprocessing and MaxHS. Our empirical results on standard crafted and industrial weighted partial MaxSAT Evaluation benchmarks show overall improvements over the original MaxHS algorithm both with and without SAT-based preprocessing.

1 Introduction

Boolean satisfiability (SAT) solving is a modern success story of computer science, providing means of solving various types of hard computational problems, based on both direct applications of SAT solvers, as well as on using SAT solvers as core NP procedures within more complex decision and optimization procedures. This success is based on several breakthroughs in practical solver techniques, central to which is preprocessing [Eén and Biere, 2005; Heule *et al.*, 2010; Järvisalo *et al.*, 2012]. However, applying SAT-level preprocessing in more complex applications of SAT solvers, such as minimal unsatisfiable core extraction [Belov *et al.*, 2013a], maximum satisfiability [Belov *et al.*, 2013b], and model counting [Lagniez and Marquis, 2014], becomes more difficult, as many of the central SAT preprocessing techniques can no longer be applied directly without losing correctness.

In this work, we focus on the Maximum satisfiability (MaxSAT) problem [Li and Manyà, 2009; Morgado *et al.*, 2013; Ansótegui *et al.*, 2013], a well-known optimization variant of SAT. Due to recent progress in MaxSAT solving [Heras *et al.*, 2011; Koshimura *et al.*, 2012; Davies and Bacchus, 2013a; 2013b; Morgado *et al.*, 2013; Ansótegui and Gabàs, 2013; Ansótegui *et al.*, 2013; Morgado *et al.*, 2014; Martins *et al.*, 2014], MaxSAT finds an increasing number of applications today [Jose and Majumdar, 2011; Zhu *et al.*, 2011; Guerra and Lynce, 2012; Berg and Järvisalo, 2013; Berg *et al.*, 2014; Bunte *et al.*, 2014]. While some of the most important SAT preprocessing techniques, such as bounded variable elimination [Eén and Biere, 2005], cannot be directly applied in the context of MaxSAT [Belov *et al.*, 2013b], a workaround is provided by applying the so-called labelled CNF (LCNF) framework [Belov and Marques-Silva, 2012].

We focus on improving the performance of the MaxHS approach [Davies and Bacchus, 2011; 2013a; 2013b] to MaxSAT solving via SAT-based preprocessing. MaxHS implements a hybrid approach to MaxSAT based on alternating between SAT-based unsatisfiable core extraction and integer programming (IP) based optimal hitting set computation over the unsatisfiable cores. The solver was one of the best in the 2014 MaxSAT Evaluation in the crafted weighted partial MaxSAT category. Motivated by this, we develop a lifting of MaxHS that works directly on LCNFs for solving MaxSAT instances, which allows for a tight integration of SAT-based preprocessing and MaxHS, and specifically, allows for *directly re-using* assumption variables from the SAT-based preprocessing step within the MaxHS solver loop. MaxHS computation heavily relies on assumption variables (both in the SAT solver and the IP solver), enabling more re-use of assumption variables from the preprocessing phase during the whole execution of the solver compared to e.g. earlier work on integrating preprocessing with MaxSAT algorithms [Belov *et al.*, 2013b]. The re-use is beneficial in terms of both having to introduce less clauses to the solver, and, as we explain, enabling more inference within MaxHS. We formally prove the correctness of the proposed LCNF-level lifting of MaxHS, and present details on how the lifting can be realized by minor modifications to the original MaxHS implementation. We present empirical results using our own competitive re-implementation of MaxHS, with additional features for implementing the LCNF-level lifting of MaxHS. The results show the benefits of the tighter integration of preprocessing and MaxHS, with overall improvements over the original MaxHS algorithm both with and without SAT-based preprocessing, on standard crafted and industrial weighted partial MaxSAT Evaluation benchmarks.

2 SAT, Preprocessing, and MaxSAT

SAT. For a Boolean variable x, there are two literals, x and $\neg x$. A clause is a disjunction ($\vee$) of literals. A truth assignment is a function from Boolean variables to $\{0,1\}$. A clause C is satisfied by a truth assignment τ ($\tau(C) = 1$) if $\tau(x) = 1$ for a literal x in C, or $\tau(x) = 0$ for a literal $\neg x$ in C. A set $F = \{C_1, \ldots, C_m\}$ of clauses, or equivalently, the conjunctive normal form (CNF) formula $\bigwedge_{i=1}^{m} C_i$, is satisfiable ($F \in$ SAT) if there is an assignment τ satisfying all clauses in F ($\tau(F) = 1$), and unsatisfiable ($\tau(F) = 0$ for any assignment τ; $F \in$ UNSAT) otherwise. The Boolean satisfiability problem (SAT) is to decide whether a given CNF formula is satisfiable.

SAT Preprocessing. The resolution rule states that, given two clauses $C_1 = (x \lor A)$ and $C_2 = (\neg x \lor B)$, the clause $C = (A \lor B)$, the *resolvent* of C_1 and C_2, can be inferred by *resolving* on the variable x. We write $C = C_1 \bowtie_x C_2$. This is lifted to two sets S_x and $S_{\neg x}$ of clauses that all contain the literal x and $\neg x$, resp., by $S_x \bowtie_x S_{\neg x} = \{C_1 \bowtie_x C_2 \mid C_1 \in S_x, C_2 \in S_{\neg x}$, and $C_1 \bowtie_x C_2$ is not a tautology$\}$.

Bounded variable elimination (VE) [Eén and Biere, 2005], currently the most important SAT preprocessing technique, follows the Davis-Putnam procedure (DP). The elimination of a variable x in a CNF formula is computed by resolving pairwise each clause in S_x with every clause in $S_{\neg x}$. Replacing the original clauses in $S_x \cup S_{\neg x}$ with the non-tautological resolvents $S = S_x \bowtie_x S_{\neg x}$ gives the CNF $(F \setminus (S_x \cup S_{\neg x})) \cup S$ that is equisatisfiable with F. To avoid exponential space complexity, VE is bounded typically by requiring that a variable x can be eliminated only if the resulting CNF formula $(F \setminus (S_x \cup S_{\neg x})) \cup S$ will not contain more than Δ more clauses than the original formula F [Eén and Biere, 2005].

A clause C in a CNF formula F is subsumed if there is a clause $C' \subset C$ in F. *Subsumption elimination* (SE) removes subsumed clauses. The *self-subsuming resolution* rule states that, given two clauses $C, D \in F$ such that (i) $l \in C$ and $\neg l \in D$ for some literal l, and (ii) D is subsumed by $C \bowtie_l D$, D can be *replaced* with $C \bowtie_l D$ in F (or, informally, $\neg l$ can be removed from D). A step of self-subsuming resolution (SSR), resolving C and D on l, gives the formula $(F \setminus D) \cup \{C \bowtie_l D\}$.

A clause C of a CNF formula F is *blocked* [Kullmann, 1999] if there is a literal $l \in C$ such that for every clause $C' \in F$ with $\neg l \in C'$, the resolvent $(C \setminus \{l\}) \cup (C' \setminus \{\neg l\})$ obtained from resolving C and C' on l is a tautology. *Blocked clause elimination* (BCE) [Järvisalo *et al.*, 2010] removes blocked clauses.

Maximum Satisfiability. An instance $F = (F_h, F_s, c)$ of the *weighted partial MaxSAT* problem consists of a set F_h of *hard* clauses, a set F_s of *soft* clauses, and a function $c : F_s \to \mathbb{N}$ that associates a non-negative cost (weight) with each of the soft clauses. Any truth assignment τ that satisfies F_h is a *solution* to F. The *cost* of a solution τ to F is

$$\textsc{cost}(F, \tau) = \sum_{C \in F_s} (1 - \tau(C)) \cdot c(C),$$

i.e., the sum of the costs of the soft clauses not satisfied by τ. A solution τ is (globally) *optimal* for F if $\textsc{cost}(F, \tau) \leq \textsc{cost}(F, \tau')$ holds for any solution τ' to F. The cost of the optimal solutions of F is denoted by $\textsc{opt}(F)$. Given a weighted partial MaxSAT instance F, the weighted partial MaxSAT problem asks to find an optimal solution to F. From here on, we refer to weighted partial MaxSAT instances simply as MaxSAT instances.

An *unsatisfiable core* of a MaxSAT instance $F = (F_h, F_s, c)$ is a subset $F'_s \subseteq F_s$ such that $F_h \cup F'_s \in$ UNSAT. An unsatisfiable core F'_s is *minimal* (MUS) if $F_h \cup F''_s \in$ SAT for all $F''_s \subset F'_s$.

3 Labelled CNFs and MaxSAT

The framework of *labelled CNFs* (LCNFs) [Belov and Marques-Silva, 2012; Belov *et al.*, 2013b] allows for generalizing MaxSAT into maximum satisfiability of LCNF, as well as for lifting SAT preprocessing techniques to MaxSAT. Assume a countable set of labels Lbl. A labelled clause C^L consists of a clause C and a (possibly empty) set of labels $L \subseteq Lbl$. A LCNF formula Φ is a set of labelled clauses. We use $Cl(\Phi)$ and $Lbls(\Phi)$ to denote the set of clauses and labels of Φ, respectively. A LCNF formula is satisfiable iff $Cl(\Phi)$ (which is a CNF formula) is satisfiable.

Given a LCNF formula Φ and a subset of its labels $M \subset Lbls(\Phi)$, the subformula $\Phi|_M$ of Φ induced by M is the LCNF formula $\{C^L \in \Phi : L \subset M\}$, i.e., the subformula obtained by removing from Φ all labelled clauses with at least one label not in M. An *unsatisfiable core* of an unsatisfiable LCNF formula Φ is a label-set $L \subset Lbls(\Phi)$ such that (i) the formula $\Phi|_L$ is unsatisfiable, and (ii) if the formula $\Phi|_{L'}$ is satisfiable for all $L' \subset L$, then L is an LMUS. We denote the set of minimal unsatisfiable cores (LMUSes) of Φ by LMUS(Φ) A *minimal correction subset* (MCS) for Φ is a label-set $R \subset Lbls(\Phi)$ such that (i) the formula $\Phi|_{Lbls(\Phi) \setminus R}$ is satisfiable, and (ii) the formula $\Phi|_{Lbls(\Phi) \setminus R'}$ is unsatisfiable for all $R' \subset R$.

In a *weighted* LCNF formula Φ, a positive weight w_i is associated with each label in $Lbls(\Phi)$. The cost of a label-set $L \subset Lbls(\Phi)$ is the sum of the weights of labels in L. Given a weighted LCNF formula Φ such that $\Phi|_\emptyset$ is satisfiable, any assignment τ that satisfies $\Phi|_\emptyset$ is a solution to the MaxSAT problem of LCNF formulas. A solution τ is optimal if it satisfies $\Phi|_{Lbls(\Phi) \setminus R}$ for some minimum-cost MCS R of Φ. The cost of τ is the cost of R.

From MaxSAT to Weighted LCNF MaxSAT. A MaxSAT instance $F = (F_h, F_s, c)$ can viewed as a weighted LCNF MaxSAT instance Φ_F by introducing (i) for each hard clause $C \in F_h$ the labelled clause $C^\emptyset$, and (ii) for each soft clause $C \in F_s$ the labelled clause $C^{\{l_C\}}$, where l_C is a distinct label for C with weight $c(C)$. It is easy to see that any optimal solution to Φ_F is an optimal solution to F, and vice versa.

From Weighted LCNF MaxSAT to MaxSAT. A *direct encoding* [Belov *et al.*, 2013b] of a weighted LCNF MaxSAT instance Φ as a MaxSAT instance F_Φ is as follows. Associate with each label $l_i \in Lbls(\Phi)$ a distinct variable a_i, and introduce (i) for each labelled clause $C^L \in \Phi$ a hard clause $C \lor \bigvee_{l_i \in L} a_i$, and (ii) for each $l_i \in Lbls(\Phi)$, a soft clause $(\neg a_i)$ with cost $c(a_i) = w_i$, where w_i is the weight of the label l_i. The resulting instance can then be input to any MaxSAT solver.

3.1 SAT Preprocessing for MaxSAT via LCNFs

Assume that we apply a SAT preprocessing technique P directly on a MaxSAT instance F, not making a distinction between the hard and soft clauses, and perhaps adjusting the weights of the clauses in the resulting MaxSAT instance in some way (weight ∞ implying a hard clause). Following [Belov *et al.*, 2013b], a SAT preprocessing technique P is *sound* for MaxSAT if there is a poly-time computable function α_P such that for any MaxSAT instance F and any optimal solution τ of $P(F)$, $\alpha_P(\tau)$ is an optimal solution of F. As argued in [Belov *et al.*, 2013b], based on the fact that blocked clause elimination does not affect the set of MUSes of any CNF formula [Belov *et al.*, 2013a], it can be shown that BCE is sound for MaxSAT. On the other hand, as shown in [Belov *et al.*, 2013b], directly applying bounded variable elimination, self-subsuming resolution, or even subsumption elimination is not sound.

As a remedy to this problem, in [Belov *et al.*, 2013b] liftings of VE, SSR, and SE to LCNF formulas were proposed. Essentially, the techniques can be applied on LCNFs by taking into account the natural restrictions implied by the SAT-level techniques on the label-sets of labelled clauses.

- The resolution rule is lifted to labelled clauses by defining the resolvent $(x \vee A)^{L_1} \bowtie_x (\neg x \vee B)^{L_2}$ of two labelled clauses $(x \vee A)^{L_1}$ and $(\neg x \vee B)^{L_2}$ as $(A \vee B)^{L_1 \cup L_2}$. The rule is lifted to two sets Φ_1 and Φ_2 of labelled clauses analogously to the CNF case.
- Eliminating a variable x then gives the LCNF $(\Phi \setminus (\Phi_x \cup \Phi_{\neg x})) \cup \Phi_x \bowtie_x \Phi_{\neg x}$, resulting a natural lifting of *bounded variable elimination for LCNFs*.
- The *self-subsuming resolution rule for LCNFs*, given two labelled clauses $C_1^{L_1} = (x \vee A)^{L_1}$ and $C_2^{L_2} = (\neg x \vee B)^{L_2}$ such that $A \subset B$ *and* $L_1 \subseteq L_2$ results in the formula $(\Phi \setminus \{C_2^{L_2}\}) \cup B^{L_2}$.
- A labelled clause $C_1^{L_1}$ subsumes $C_2^{L_2}$ if both $C_1 \subset C_2$ *and* $L_1 \subseteq L_2$, which gives the redundancy property used for *subsumption elimination for LCNFs*.

Here it is important to notice that, due to the resolution rule for LCNFs, bounded variable elimination and self-subsuming resolution can cause an increase in the size of the label-sets of the resulting labelled clauses. In particular, consider the encoding of MaxSAT as weighted LCNF MaxSAT. Even though each labelled clause corresponding to a soft clause in the original MaxSAT instance will have a singleton label-set, after LCNF-level preprocessing some of the clauses can have label-sets with more than one label. Direct encoding of the preprocessed weighted LCNF MaxSAT instance as a MaxSAT instance will then add multiple new variables, corresponding to the labels of the labelled clauses, to the resulting soft clauses. Furthermore, we note that in some cases LCNF-level preprocessing may result in a labelled clause $\emptyset^L$, i.e., a labelled clause the actual clause of which is empty.

We further note that, when applying BCE together with VE, SSE, and SE, it makes sense to consider a straightforward lifting of BCE to LCNF formulas to simplify the preprocessing pipeline: a labelled clause C^L is blocked in a LCNF formula Φ if C is blocked in $Cl(\Phi)$. As BCE is sound for MaxSAT, it clear that this lifting is sound for LCNF MaxSAT.

4 Lifting MaxHS to Weighted LCNFs

As discussed in [Belov *et al.*, 2013b], SAT-based preprocessing can be applied in the context of MaxSAT solving using the following observations. (i) By viewing MaxSAT instances as weighted LCNF MaxSAT instances, the LCNF-liftings of VE, SSR, and SE can be soundly applied on MaxSAT instances; and (ii) using the reduction from weighted LCNF MaxSAT to MaxSAT, one can directly employ any MaxSAT solver to obtain solutions to preprocessed weighted LCNF MaxSAT instances, and hence also to the original MaxSAT instances. However, part (ii) in this flow can in cases be non-optimal, especially when applying one of the many SAT-based MaxSAT solvers which use *assumptions* for switching on and off soft clauses from one SAT solver call to another.[1]

An alternative, as done in this work, is to develop liftings of the SAT-based MaxSAT solvers for weighted LCNF MaxSAT. A benefit of doing so is that such liftings can *use the labels of the labelled clauses directly as assumption variables for the SAT solver calls.* By doing this, one avoids both adding the additional soft unit clauses introduced by the direct encoding from weighted LCNF MaxSAT to MaxSAT, as well as the *additional layer of assumption variables* added afterwards by the MaxSAT solver to the soft clauses.

The special nature of preprocessed MaxSAT instances—assumption variables being distributed to multiple clauses, and individual clauses having multiple assumption variables—requires care in how the assumptions are used, which depends on the SAT-based MaxSAT algorithm being lifted to weighted LCNF MaxSAT. In this work we lift the previously proposed MaxHS [Davies and Bacchus, 2011; 2013a; 2013b] algorithm into the LCNF framework. Our motivation for this is that MaxHS—one of the best-performing solvers in the 2014 MaxSAT Evaluation crafted weighted partial category—heavily relies on assumption variables. This makes it a prime candidate for integrating the idea of re-using assumption variables from the preprocessing phase.

MaxHS. An overview of MaxHS is presented as Algorithm 1. MaxHS is a core-guided algorithm that exploits the fact that, when invoked on an unsatisfiable set of clauses, most CDCL SAT solvers can output an unsatisfiable core over the assumption variables used in the solver calls. During execution, MaxHS maintains a collection $\mathcal{C}$ of cores (over the soft clauses) of the input MaxSAT instance $F = (F_h, F_s, c)$. At each iteration, a minimum-cost hitting set H over $\mathcal{C}$ is computed. This hitting set problem is stated over the assumption variables and solved using an IP solver. A SAT solver is then invoked on $F_h \wedge F_s$ with the assumption variables in H set to 1 (and the other assumption variables to 0). If the solver reports *satisfiable*, the algorithm terminates and returns the truth assignment produced by the SAT solver, which is guaranteed to be an optimal solution to F. If the solver reports

[1]Assumptions refer to adding a distinct fresh variable a_i to each of the soft clauses C_i in the input formula. Calling the SAT solver under the assumption $a_i = 1$ is equivalent to *removing* C_i from the instance. Similarly, the assumption $a_i = 0$ switches the clause on.

```
Input: A MaxSAT instance $F = (F_h, F_s, c)$
Output: An optimal solution $\tau$ for $F$
$\mathcal{C} \leftarrow \emptyset$                    // set of found unsat cores of $F$
while true do
    $H \leftarrow$ MinCostHittingSet($\mathcal{C}$)
    $(result, C, \tau) \leftarrow$ SATSolve($F_h \cup (F_s \setminus H)$)
    if result="satisfiable" then
        return $\tau$                             // solver returned SAT
    else
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$   // solver returned unsat core of $F$
    end
end
```

Algorithm 1: The MaxHS algorithm

unsatisfiable, the algorithm obtains a new core C from the SAT solver and reiterates. The intuition behind the algorithm is that when the reduced formula is satisfiable, the found hitting set is also a minimum-cost hitting set over *all* MUSes of F. Hence removing the clauses in the hitting set removes all sources of unsatisfiability from the formula in a minimum-cost manner [Davies and Bacchus, 2011].

MaxHS for Weighted LCNFs. While our lifting of the MaxHS algorithm to LCNFs, LCNF-MaxHS (Alg. 2), closely follows the original MaxHS algorithm, it also makes a critical shift from the clause-centric view (with a single distinct assumption variable for each soft clause) to a label-centric view in which overlapping label-sets with more than one label are allowed. This generalizes MaxHS to LCNFs, while still maintaining correctness (as proven in the following). The unsatisfiable cores on the LCNF-level are explicitly maintained as sets of labels. On each iteration, LCNF-MaxHS checks the satisfiability of the subformula *now induced by* $Lbls(\Phi) \setminus R$ for some minimum-cost hitting set over the collection of identified cores $\mathcal{L}$ of Φ. Notice that inducing a subformula by $Lbls(\Phi) \setminus R$ is analogous to removing all clauses present in the hitting set of the original MaxHS algorithm.

4.1 Correctness

We proceed by a formal correctness proof for LCNF-MaxHS, which relies on the hitting set duality theorem for LCNFs [Belov and Marques-Silva, 2012]. Recall that a hitting set H over an arbitrary collection of sets $\mathcal{S}$ is *irreducible* if no $H' \subset H$ is a hitting set over $\mathcal{S}$.

Theorem 1 *A label-set $M \subset Lbls(\Phi)$ of a LCNF formula Φ is an MCS of Φ iff it is an irreducible hitting set over* LMUS(Φ).

The correctness follows from the following.

Proposition 1 *Let Φ be a LCNF formula, $\mathcal{L} \subset \mathcal{P}(Lbls(\Phi))$ a set of its cores, and R a minimum-cost hitting set over $\mathcal{L}$. Assume τ is an assignment satisfying $\Phi|_{Lbls(\Phi)\setminus R}$. Then R is a minimum-cost irreducible hitting set over* LMUS(Φ).

Proof. 1) R is a hitting set over LMUS(Φ). Otherwise there would be a LMUS M of Φ such that $M \subset Lbls(\Phi) \setminus R$, contradicting the assumption that $\Phi|_{Lbls(\Phi)\setminus R}$ is satisfiable.

2) R is irreducible as any $R' \subset R$ that is a hitting set over LMUS(Φ) is also a hitting set over $\mathcal{L}$. As R' contains fewer

```
Input: A weighted LCNF MaxSAT instance $\Phi$
Output: An optimal solution $\tau$ for $\Phi$
$\mathcal{L} \leftarrow \emptyset$              // set of found unsat cores of $Lbls(\Phi)$
while true do
    $R \leftarrow$ MinCostHittingSet($\mathcal{L}$)
    $(result, L, \tau) \leftarrow$ SATSolve($\Phi|_{Lbls(\Phi)\setminus R}$)
    if result="satisfiable" then
        return $\tau$                        // solver returned SAT
    else
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{L\}$      // solver returned unsat core of
                                          $Lbls(\Phi)$
    end
end
```

Algorithm 2: LCNF-MaxHS, lifting of MaxHS to LCNFs

labels than R, it has to be of lower cost, contradicting the assumed minimum cost (over the hitting sets of $\mathcal{L}$) of R.

3) R has minimum cost over all hitting sets of LMUS(Φ) which follows, similarly to case 2, from the fact that any hitting set R'' over LMUS(Φ) is also a hitting set over $\mathcal{L}$. Hence R'' has to have at least the same cost as R. □

Theorem 2 *The assignment τ returned by the LCNF-MaxHS algorithm is an optimal solution to the weighted MaxSAT problem for LCNFs.*

Proof. By Proposition 1, τ satisfies $\Phi|_{Lbls(\Phi)\setminus R}$ for a minimum-cost irreducible hitting set R over LMUS(Φ). By Theorem 1, R is also a minimum-cost MCS of Φ. □

4.2 Integrating SAT-Based Preprocessing and LCNF-MaxHS

Given a MaxSAT instance $F = (F_h, F_s, c)$ as input, the discussed SAT-based preprocessing can be integrated with the lifting of MaxHS to the weighted LCNF setting as follows.

1. Apply the labelled liftings of BCE, VE, SSR, and SE on Φ_F (i.e., F as a weighted LCNF MaxSAT instance), to obtain the preprocessed LCNF Φ'_F.
2. Solve Φ'_F using LCNF-MaxHS.

In practice, the steps above can be implemented, based on the correctness of the LCNF-MaxHS algorithm, by extending the MaxHS algorithm to take as part of the input an explicit listing of assumption variables and modifying the implementation to directly use these assumption variables instead of instrumenting the input soft clauses with new assumption variables. More precisely, we do the following:

1'. Extend each $C_i \in F_s$ with a distinct new assumption variable and apply BCE, VE, SSR, and SE on $F_h \wedge \bigwedge_{C_i \in F_s}(C_i \vee a_i)$, forbiding the removal of any a_i variables during preproccessing. Divide the resulting set of clauses into (i) "hard" clauses F'_h which do not include any of the assumption variables a_i and (ii) "soft" clauses F'_s which each contain *at least one* of the assumption variables.

2'. Apply MaxHS on the MaxSAT instance (F'_h, F'_s, c'), where $c'(a_i) = c(C_i)$ for each $C_i \in F_s$, and explicitly guide MaxHS to work on the a_i variables as the assumption variables.

Notice especially that step 2' avoids adding the soft unit clauses over the assumption variables—produced by the earlier mentioned *direct encoding*—that encode the weights of the clauses to which the assumption variables are added in the direct encoding. This makes a difference when applying the MaxHS algorithm, as explained in the following.

Eq-seeding was proposed in [Davies and Bacchus, 2013a] for improving the efficiency of solving the minimum-cost hitting set problems with IP. In short, eq-seeding uses the fact that each binary clause $(l \lor a_i)$, where a_i is an assumption variable, can actually be viewed as the logical equivalence $l \leftrightarrow \neg a_i$ [Davies and Bacchus, 2013a]. While these logical equivalences are *not* added to the SAT solver, they can be used for deriving additional linear constraints that are added to the hitting set IPs as follows: if for each literal l_j of a clause $C = (l_1 \lor \cdots \lor l_m \lor l_{m+1} \lor \cdots \lor l_n)$ in the MaxSAT instance, either (i) l_j or $\neg l_j$ is equivalent to an assumption variable a_i (i.e., $(l_j \leftrightarrow a_i)$ or $(\neg l_j \leftrightarrow a_i)$); or (ii) l_j is an assumption variable itself (in which case we implicitly have $(l_j \leftrightarrow l_j)$), then replacing l_j by its equivalent assumption variable for each of the variables in C gives a linear at-least-one constraint purely over the assumption variables. These derived linear constraints are added to the IP solver.

An interesting observation here is that eq-seeding within our LCNF-MaxHS implementation can in some cases derive more linear constraints than when invoking the original MaxHS algorithm on the direct encoding after preprocessing.

Example 1 *Assume that, after preprocessing, we have the LCNF formula* $\Phi = \{(\emptyset)^{\{a_1,a_2\}}, (\neg x_1)^{\{a_3\}}, (\neg x_2)^{\{a_4\}}, (x_1 \lor x_2)^{\emptyset}\}$. *For LCNF-MaxHS, these labelled clauses are represented as the clauses* $F = \{(a_1 \lor a_2), (\neg x_1 \lor a_3), (\neg x_2 \lor a_4), (x_1 \lor x_2)\}$ *where the* a_i*s are used as assumption variables. From* $(a_1 \lor a_2)$, *eq-seeding infers the linear constraint* $a_1 + a_2 \geq 1$. *Furthermore, since* a_3 *can be considered equivalent to* x_1, *and* a_4 *to* x_2, *eq-seeding infers* $a_3 + a_4 \geq 1$. *In contrast, consider invoking the original MaxHS algorithm on the direct encoding of* Φ, *i.e., the MaxSAT instance* (F_h, F_s) *with* $F_h = F$ *and* $F_s = \{(\neg a_1), (\neg a_2), (\neg a_3), (\neg a_4)\}$. *Without any knowledge of the fact that the* a_i *variables could be used as assumptions, MaxHS will add to each unit soft clause* $(\neg a_i)$ *a new assumption variable* b_i, *giving the clause* $(\neg a_i \lor b_i)$, *and will hence consider* a_i *to be equivalent to* b_i *for each* i. *From this, eq-seeding can still infer* $b_1 + b_2 \geq 1$, *which is equivalent to* $a_1 + a_2 \geq 1$ *inferred by LCNF-MaxHS. However, eq-seeding in MaxHS will not be able to infer the second linear constraint inferred by eq-seeding within LCNF-MaxHS.*

Special Cases Arising from Preprocessing. Finally, we note an interesting special case arising purely from applying the labelled preprocessing techniques to MaxSAT instances. As already mentioned, in our experiments we often observed labelled clauses of the form $\emptyset^L$, which is equivalent to a MaxSAT clause $\bigvee_{l_i \in L} a_i$, i.e., a clause consisting purely of assumption variables. In fact, in the experiments we report on in the following, we observed that for some benchmarks, preprocessing resulted in instances purely consisting of such clauses. Such clauses can be directly added as the linear constraint $\sum_{l_i \in L} a_i \geq 1$ to the IP solver used for solving the hitting set problems. This is actually done automatically by the eq-seeding technique proposed in [Davies and Bacchus, 2013a] and implemented in our solver.

5 Implementation and Experiments

For implementing the lifting of MaxHS to weighted LCNFs (Alg. 2), we extended our own prototype re-implementation of the MaxHS algorithm for weighted LCNF MaxSAT. Refining Algorithm 1, this re-implementation includes the SAT solver tweaks and disjoint phase of [Davies and Bacchus, 2011], the non-optimal hitting set computations of [Davies and Bacchus, 2013b], as well as the core minimization and eq-seeding techniques of [Davies and Bacchus, 2013a]. MiniSAT 2.2.0 [Eén and Sörensson, 2003] is used as the underlying SAT solver and IBM CPLEX 12.6.0 [Cpl, 2015] is used to solve the minimum-cost hitting set IPs. We extended our MaxHS implementation to take a list of assumption variables as part of the solver input.

As the SAT preprocessor, we used Coprocessor 2.0 [Manthey, 2012] that we modified to add the required assumption variables to the soft clauses, and used its whitelisting feature to forbid removal of any occurrences of the assumption variables during preprocessing. For the experiments reported on, we did not yet integrate Coprocessor into our MaxHS re-implementation. Instead, although somewhat non-optimal in terms of time spent on preprocessing, we called Coprocessor sepaately from the solver, applying BCE, VE, SSR, and SE. The total preprocessing time is included in the running times reported.

We report on experiments using the following solvers.

MHS2.5: The most recent version of the original implementation of the MaxHS algorithm (reference purposes).
MHS: our re-implementation of MaxHS.
MHS+pre: MHS with preprocessing, using the direct encoding after preprocessing.
LMHS+pre: MHS with preprocessing, re-using the assumption variables from preprocessing (i.e., LCNF-MaxHS with preprocessing).
Eva: the best-performing solver in the weighted partial industrial category of MaxSAT Evaluation 2014 [Narodytska and Bacchus, 2014].
Eva+pre: Eva with preprocessing, using the direct encoding after preprocessing.

We used all 624 instances from the weighted partial crafted (214) and industrial (410) categories of MaxSAT Evaluation 2014 (http://www.maxsat.udl.cat/14/). Note that the weighted partial crafted benchmark set contains 310 instances; however, 96 of the instances do not contain hard clauses. The experiments were run on a cluster of 2.53-GHz Intel Xeon quad core machines with 32-GB memory and Ubuntu Linux 12.04. A timeout of 1 h was enforced for solving each benchmark instance.

Results are presented in Figures 1 and 2. Figure 2 shows the number of instances solved for different time limits over all the benchmarks. For example, to solve 500 instances, MHS needs a per-instance timeout of 1500 s, while less than

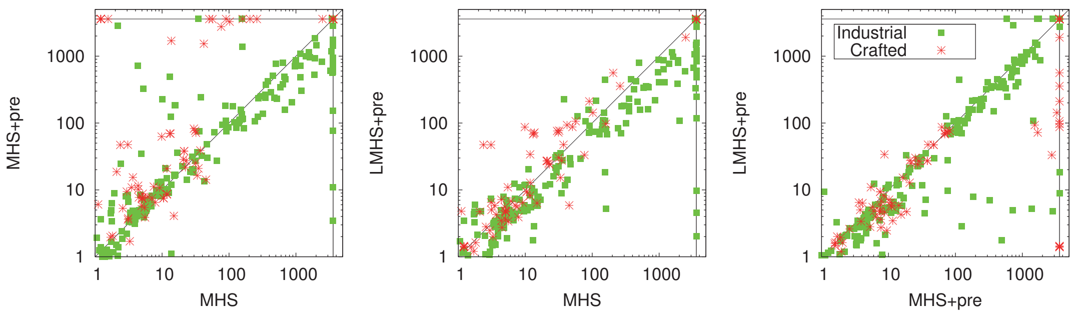

Figure 1: Comparison of MaxHS variants with and without preprocessing, runtimes in seconds. Left: MaxHS w/o preprocessing v MaxHS w/preprocessing using the direct encoding; middle: MaxHS w/o preprocessing v LCNF-MaxHS w/preprocessing; right: MaxHS w/preprocessing using the direct encoding v LCNF-MaxHS w/preprocessing.

500 s suffices for LMHS+pre. Using the direct encoding after preprocessing decreases the performance of MHS, especially on the crafted instances. LMHS+pre clearly improves over the direct encoding and over not using preprocessing at all. Also note that our re-implementation of MaxHS appears to be competitive when compared to the latest version of the original MaxHS solver (MHS2.5), as well as Eva when comparing over all weighted partial instances. Figure 2 also gives some insight into the effect of the individual preprocessing techniques on the performance of LMHS. We ran two sets of experiments, one only using VE, SSR, and SE (“NoBCE”) and one only using BCE (“OnlyBCE”). The combination of all techniques resulted in 99 timeouts for LMHS, while using BCE only resulted in 98 and leaving out BCE in 95 timeouts. These differences are due to industrial instances.

Figure 1 gives a pairwise comparison of MHS, MHS+pre, and LMHS+pre. Figure 1 (right) shows that LMHS+pre (preprocessing and re-using assumptions) improves noticeably on MHS+pre (preprocessing and direct encoding). MHS+pre performs noticeably worse than MHS (no preprocessing) on the crafted instances (Figure 1 left). LMHS+pre improves on MHS+pre on these instances (Figure 1 right), with performance closer to that of MHS. On the industrial instances, the results between MHS+pre and MHS are inconclusive. LMHS+pre on the other hand improves somewhat on MHS+pre and more on MHS. LMHS+pre timed out on 99 instances (18 crafted, 81 industrial), MHS on 111 (18, 93). LMHS+pre timed out on only 1 (industrial) instance solved by MHS, while MHS on 13 instances solved by LMHS+pre (all industrial). We also conducted experiments on the unweighted partial crafted benchmarks from the 2014 MaxSAT Evaluation: MHS timed out on 212, (88 of 421 crafted and 124 of 568 industrial), MHS+pre on 280 (129, 151) and LMHS+pre on 204 (84, 120). As a comparison, MHS2.5 timed out on 200 instances (90, 110) and Open-WBO [Martins *et al.*, 2014], one of the best-performing solvers in the 2014 partial industrial track, on 206 (122, 84).

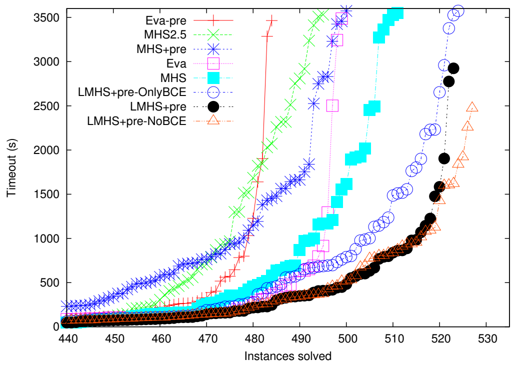

Figure 2: Cactus plot comparing the different MHS variants

6 Conclusions

We presented a lifting of the MaxHS algorithm to labelled LCNFs, enabling a tighter integration of preprocessing and MaxHS via re-using assumption variables from the preprocessing step. We explained how the lifting can be implemented via modifications to MaxHS, and pointed out concrete examples of why assumption re-use can be beneficial. Experiments showed that our LCNF lifting of MaxHS does improve the effectiveness of preprocessing especially on crafted weighted partial MaxSAT, and also improves on the overall performance of MaxHS both with and without direct preprocessing. For future work, an interesting question is if other MaxSAT solvers, such as Eva, could benefit from tighter integration of preprocessing.

Acknowledgments

Work funded by Academy of Finland, grants 251170 COIN Centre of Excellence in Computational Inference Research, 276412, and 284591; and Research Funds of the University of Helsinki.

References

[Ansótegui and Gabàs, 2013] C. Ansótegui and J. Gabàs. Solving (weighted) partial MaxSAT with ILP. In *Proc. CPAIOR*, volume 7874 of *LNCS*, pages 403–409. Springer, 2013.

[Ansótegui *et al.*, 2013] C. Ansótegui, M.L. Bonet, and J. Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[Belov and Marques-Silva, 2012] A. Belov and J. Marques-Silva. Generalizing redundancy in propositional logic: Foundations and hitting sets duality. *CoRR*, abs/1207.1257, 2012.

[Belov *et al.*, 2013a] A. Belov, M. Järvisalo, and J. Marques-Silva. Formula preprocessing in MUS extraction. In *Proc. TACAS*, volume 7795 of *LNCS*, pages 108–123. Springer, 2013.

[Belov *et al.*, 2013b] A. Belov, A. Morgado, and J. Marques-Silva. SAT-based preprocessing for MaxSAT. In *Proc. LPAR-19*, volume 8312 of *LNCS*, pages 96–111. Springer, 2013.

[Berg and Järvisalo, 2013] J. Berg and M. Järvisalo. Optimal correlation clustering via MaxSAT. In *Proc. 2013 IEEE ICDM Workshops*, pages 750–757. IEEE Press, 2013.

[Berg *et al.*, 2014] J. Berg, M. Järvisalo, and B. Malone. Learning optimal bounded treewidth bayesian networks via maximum satisfiability. In *Proc. AISTATS*, volume 33, pages 86–95. JMLR, 2014.

[Bunte *et al.*, 2014] Kerstin Bunte, Matti Järvisalo, Jeremias Berg, Petri Myllymäki, Jaakko Peltonen, and Samuel Kaski. Optimal neighborhood preserving visualization by maximum satisfiability. In *Proc. AAAI*, pages 1694–1700. AAAI Press, 2014.

[Cpl, 2015] CPLEX, 2015. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[Davies and Bacchus, 2011] J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proc. CP*, volume 6876 of *LNCS*, pages 225–239. Springer, 2011.

[Davies and Bacchus, 2013a] J. Davies and F. Bacchus. Exploiting the power of MIP solvers in MaxSAT. In *Proc. SAT*, volume 7962 of *LNCS*, pages 166–181. Springer, 2013.

[Davies and Bacchus, 2013b] J. Davies and F. Bacchus. Postponing optimization to speed up MAXSAT solving. In *Proc. CP*, volume 8124 of *LNCS*, pages 247–262. Springer, 2013.

[Eén and Biere, 2005] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proc. SAT*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.

[Eén and Sörensson, 2003] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[Guerra and Lynce, 2012] J. Guerra and I. Lynce. Reasoning over biological networks using maximum satisfiability. In *Proc. CP*, volume 7514 of *LNCS*, pages 941–956. Springer, 2012.

[Heras *et al.*, 2011] F. Heras, A. Morgado, and J. Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *Proc. AAAI*. AAAI Press, 2011.

[Heule *et al.*, 2010] M. Heule, M. Järvisalo, and A. Biere. Clause elimination procedures for CNF formulas. In *Proc. LPAR-17*, volume 6397 of *LNCS*, pages 357–371. Springer, 2010.

[Järvisalo *et al.*, 2010] M. Järvisalo, A. Biere, and M. Heule. Blocked clause elimination. In *Proc. TACAS*, volume 6015 of *LNCS*, pages 129–144. Springer, 2010.

[Järvisalo *et al.*, 2012] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In *Proc. IJCAR*, volume 7364 of *LNCS*, pages 355–370. Springer, 2012.

[Jose and Majumdar, 2011] M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Proc. PLDI*, pages 437–446. ACM, 2011.

[Koshimura *et al.*, 2012] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal of Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012.

[Kullmann, 1999] O. Kullmann. On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149–176, 1999.

[Lagniez and Marquis, 2014] J.-M. Lagniez and P. Marquis. Preprocessing for propositional model counting. In *Proc. AAAI*, pages 2688–2694. AAAI Press, 2014.

[Li and Manyà, 2009] C.M. Li and F. Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, pages 613–631. IOS Press, 2009.

[Manthey, 2012] N. Manthey. Coprocessor 2.0 - A flexible CNF simplifier. In *Proc. SAT*, volume 7317 of *LNCS*, pages 436–441. Springer, 2012.

[Martins *et al.*, 2014] R. Martins, S. Joshi, V.M. Manquinho, and I. Lynce. Incremental cardinality constraints for MaxSAT. In *Proc. CP*, volume 8656 of *LNCS*, pages 531–548. Springer, 2014.

[Morgado *et al.*, 2013] A. Morgado, F. Heras, M.H. Liffiton, J. Planes, and J. Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[Morgado *et al.*, 2014] A. Morgado, C. Dodaro, and J. Marques-Silva. Core-guided maxsat with soft cardinality constraints. In *Proc. CP*, volume 8656 of *LNCS*, pages 564–573. Springer, 2014.

[Narodytska and Bacchus, 2014] N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proc. AAAI*, pages 2717–2723, 2014.

[Zhu *et al.*, 2011] C.S. Zhu, G. Weissenbacher, and S. Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In *Proc. FMCAD*, pages 63–66, 2011.

Paper II

Jeremias Berg, Paul Saikko, and Matti Järvisalo

Re-using Auxiliary Variables for MaxSAT Preprocessing

2015 IEEE 27th International Conference on Tools with Artificial Intelligence

Re-using Auxiliary Variables for MaxSAT Preprocessing

Jeremias Berg and Paul Saikko and Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

***Abstract*—Solvers for the maximum satisfiability (MaxSAT) problem—a well-known optimization variant of Boolean satisfiability (SAT)—are finding an increasing number of applications. Preprocessing has proven an integral part of the SAT-based approach to efficiently solving various types of real-world problem instances. It was recently shown that SAT preprocessing for MaxSAT becomes more effective by re-using the auxiliary variables introduced in the preprocessing phase directly in the SAT solver within a core-based hybrid MaxSAT solver. We take this idea of re-using auxiliary variables further by identifying them among variables already present in the input MaxSAT instance. Such variables can be re-used already in the preprocessing step, avoiding the introduction of multiple layers of new auxiliary variables in the process. Empirical results show that by detecting auxiliary variables in the input MaxSAT instances can lead to modest additional runtime improvements when applied before preprocessing. Furthermore, we show that by re-using auxiliary variables not only within preprocessing but also as assumptions within the SAT solver of the MaxHS MaxSAT algorithm can alone lead to performance improvements similar to those observed by applying SAT-based preprocessing.**

I. Introduction

Maximum satisfiability (MaxSAT) [1], [2], [3] is a well-known optimization variant of the archetypical NP-complete problem of Boolean satisfiability (SAT). Building on the extraordinary success of SAT solvers, exact solvers for MaxSAT—and, especially, its weighted partial generalization—are finding an increasing number of applications, ranging e.g. from hardware design debugging and model-based diagnosis to bioinformatics and data analysis [4], [5], [6], [7], [8], [9], [10], [11]. This is brought on by recent improvements in MaxSAT solving techniques [12], [13], [14], [15], [2], [16], [3], [17], [18].

Recently it was shown [19] that SAT-based preprocessing [20], [21] for MaxSAT [22] can be made more effective by explicitly re-using the auxiliary variables introduced in the preprocessing phase directly as the assumption variables the SAT solver within a core-based hybrid MaxSAT solver [23], [14], [15]. In this work, we take this idea of re-using auxiliary variables further. Our idea is to automatically detect (*group detect*) auxiliary variables among the variables already present in the input MaxSAT instance. Such detected auxiliary variables can be used already within the preprocessing step. This avoids introducing layers of new auxiliary variables in the preprocessing and the solving steps. A key motivation for group detection comes from the fact that such auxiliary variables arise naturally when encoding more complex finite-domain soft constraints into MaxSAT via the so-called *Group MaxSAT* framework [24], [25].

We observe that group detection can be achieved by simple pattern matching, and show that this often identifies re-usable auxiliary variables in the weighted partial MaxSAT benchmark sets of the most recent 2014 MaxSAT Evaluation. We also detail why variable re-use via group detection can be beneficial in particular in conjunction with the MaxHS solver. We show that group detection applied before SAT-based preprocessing can bring modest runtime improvements to state-of-the-art MaxSAT solvers.

An additional benefit of group detection is that the detected auxiliary variables can be explicitly re-used as assumptions throughout the whole MaxSAT solving process—not only within SAT-based preprocessing for MaxSAT, but also in the SAT solver within SAT-based MaxSAT algorithms—by explicitly informing the MaxSAT solver of these variables. Using a recently proposed generalization of MaxHS for this purpose [19], we show that this results in overall improvements over MaxHS on weighted partial 2014 MaxSAT Evaluation benchmarks. Furthermore, surprisingly, explicitly re-using group detected variables alone results in similar overall improvements as applying SAT-based preprocessing together with variable re-use of the auxiliary variables necessary for the preprocessing phase.

The rest of the paper is organized as follows. We start with necessary preliminaries on MaxSAT and the group and labelled extensions of MaxSAT (Section II) and SAT-based preprocessing for labelled MaxSAT (Section III). We then detail the proposed approach to re-using variables present in the input MaxSAT instances via what we call group detection (Section IV). After this, we explain why group detection could be beneficial to apply in conjunction with a recently proposed labelled lifting of the MaxHS approach (Section V). Before concluding, we present results of an empirical evaluation on the effects of integrating group detection into the MaxSAT solving process (Section VI).

II. MaxSAT, Groups, and Labels

For a Boolean variable x, there are two literals, x and $\neg x$. A clause is a disjunction ($\vee$) of literals. A truth assignment is a function from Boolean variables to $\{0, 1\}$. A clause C is satisfied by a truth assignment τ ($\tau(C) = 1$) if $\tau(x) = 1$ for a literal x in C, or $\tau(x) = 0$ for a literal $\neg x$ in C. A set $F = \{C_1, \ldots, C_m\}$ of clauses, or equivalently, the conjunctive normal form (CNF) formula $\bigwedge_{i=1}^{m} C_i$, is satisfiable if there is an assignment τ satisfying all clauses in F ($\tau(F) = 1$), and unsatisfiable ($\tau(F) = 0$ for any assignment τ) otherwise. The Boolean satisfiability problem (SAT) is to decide whether a given CNF formula is satisfiable.

An instance $F = (F_h, F_s, c)$ of the *weighted partial MaxSAT* problem consists of a set F_h of *hard* clauses, a set

F_s of *soft* clauses, and a function $c : F_s \to \mathbb{N}$ that associates a non-negative cost (weight) with each of the soft clauses. Any truth assignment τ that satisfies F_h is a *solution* to F. The *cost* of a solution τ to F is $\textsc{cost}(F, \tau) = \sum_{C \in F_s}(1 - \tau(C)) \cdot c(C)$, i.e., the sum of the costs of the soft clauses not satisfied by τ. A solution τ is (globally) *optimal* for F if $\textsc{cost}(F, \tau) \leq \textsc{cost}(F, \tau')$ holds for any solution τ' to F. Given a weighted partial MaxSAT instance F, the weighted partial MaxSAT problem asks to find an optimal solution to F. From here on, we refer to weighted partial MaxSAT instances simply as MaxSAT instances. A MaxSAT instance with $c(C) = 1$ for all soft clauses C is often called *unweighted*.

An *unsatisfiable core* of a MaxSAT instance $F = (F_h, F_s, c)$ is a subset $F_s' \subseteq F_s$ such that $F_h \cup F_s'$ is unsatisfiable. An unsatisfiable core F_s' is *minimal* (an MUS) if $F_h \cup F_s'' \in \text{SAT}$ for all $F_s'' \subset F_s'$.

Group MaxSAT [24], [25] extends MaxSAT by allowing weights on (soft) *groups* of clauses. An instance $F = (F_h, \mathcal{G}_s, c)$ of weighted group MaxSAT consists of a set F_h of hard clauses, a set $\mathcal{G}_s$ of soft groups of clauses, and function $c : \mathcal{G}_s \to \mathbb{N}$ that associates a a non-negative cost with each group in $\mathcal{G}_s$. Each group $G \in \mathcal{G}_s$ is a set of clauses. A truth assignment τ satisfies G iff τ satisfies every clause in G. The Group MaxSAT problem asks to find an assignment τ that satisfies F_h and maximizes the sum of the costs of the groups satisfied by τ.

The framework of *labelled CNFs* (LCNFs) [26], [22] allows for generalizing MaxSAT and Group MaxSAT into maximum satisfiability of LCNF, as well as for lifting SAT preprocessing techniques to MaxSAT. Assume a countable set *Lbl* of labels. A labelled clause C^L consists of a clause C and a (possibly empty) set $L \subseteq \mathit{Lbl}$ of labels. An LCNF formula Φ is a set of labelled clauses. We use $\mathit{Cl}(\Phi)$ and $\mathit{Lbls}(\Phi)$ to denote the set of clauses and labels of Φ, respectively. An LCNF formula is satisfiable iff $\mathit{Cl}(\Phi)$ (which is a CNF formula) is satisfiable.

Given an LCNF formula Φ and a subset of its labels $M \subseteq \mathit{Lbls}(\Phi)$, the subformula $\Phi|_M$ of Φ induced by M is the LCNF formula $\{C^L \in \Phi : L \subseteq M\}$, i.e., the subformula obtained by removing from Φ all labelled clauses with at least one label not in M. An *unsatisfiable core* of an unsatisfiable LCNF formula Φ is a label-set $L \subseteq \mathit{Lbls}(\Phi)$ such that the formula $\Phi|_L$ is unsatisfiable. An unsatisfiable core L is a LMUS iff the formula $\Phi|_{L'}$ is satisfiable for all $L' \subset L$. A *minimal correction subset* (LMCS) for Φ is a label-set $R \subseteq \mathit{Lbls}(\Phi)$ such that (i) the formula $\Phi|_{\mathit{Lbls}(\Phi) \setminus R}$ is satisfiable, and (ii) the formula $\Phi|_{\mathit{Lbls}(\Phi) \setminus R'}$ is unsatisfiable for all $R' \subset R$.

An instance of the *weighted LCNF-MaxSAT* problem consists of an LCNF formula Φ, with a positive weight w_i associated with each label in $\mathit{Lbls}(\Phi)$. The cost of a label-set $L \subseteq \mathit{Lbls}(\Phi)$ is the sum of the weights of labels in L. Given a weighted LCNF-MaxSAT instance Φ such that $\Phi|_\emptyset$ is satisfiable, any assignment τ that satisfies $\Phi|_\emptyset$ is a solution to the MaxSAT problem of LCNF formulas. A solution τ is optimal if it satisfies $\Phi|_{\mathit{Lbls}(\Phi) \setminus R}$ for some minimum-cost LMCS R of Φ. The cost of τ is the cost of R. Similarly to MaxSAT, we will from here on refer to *weighted* LCNF-MaxSAT instances as LCNF-MaxSAT instances.

A MaxSAT instance $F = (F_h, F_s, c)$ can be viewed as a LCNF-MaxSAT instance Φ_F by introducing (i) for each hard clause $C \in F_h$ the labelled clause $C^\emptyset$, and (ii) for each soft clause $C \in F_s$ the labelled clause $C^{\{l_C\}}$, where l_C is a distinct label for C with weight $c(C)$. It is easy to see that any optimal solution to Φ_F is an optimal solution to F, and vice versa.

An LCNF-MaxSAT instance Φ can be viewed as a MaxSAT instance F_Φ [22] by associating with each label $l_i \in \mathit{Lbls}(\Phi)$ a distinct variable a_i, and introducing (i) for each labelled clause $C^L \in \Phi$ a hard clause $C \vee \bigvee_{l_i \in L} a_i$, and (ii) for each $l_i \in \mathit{Lbls}(\Phi)$, a soft clause $(\neg a_i)$ with cost $c(a_i) = w_i$, where w_i is the weight of the label l_i. We call this the *direct encoding*. Notice that converting a MaxSAT instance to LCNF and then back to MaxSAT using the direct encoding introduces new variables and clauses to the formula, as exemplified next.

Example 1: Consider the (unweighted) MaxSAT instance $F_{\text{ex}} = (F_h, F_s)$ with $F_h = \{(x \vee z), (\neg z), (y \vee z)\}$ and $F_s = \{(\neg x), (\neg y, \vee \neg z), (z \vee y), (\neg z \vee y)\}$. We will use F_{ex} as a running example in this paper. The assignment τ for which $\tau(x) = \tau(y) = 1$ and $\tau(z) = 0$ is an optimal solution to F_{ex} of cost 1. The set $\{(\neg x)\}$ is an example of a minimal unsatisfiable core of F_{ex}. The LCNF-MaxSAT instance $\Phi_{F_{\text{ex}}}$ corresponding to F_{ex} is

$$\begin{aligned}\Phi_{F_{\text{ex}}} = \{&(x \vee z)^\emptyset, (\neg z)^\emptyset, (y \vee z)^\emptyset, (\neg x)^{\{l_1\}},\\ &(\neg y, \vee \neg z)^{\{l_2\}}, (z \vee y)^{\{l_3\}}, (\neg z \vee y)^{\{l_4\}}\}.\end{aligned}$$

Now $\mathit{Cl}(\Phi_{F_{\text{ex}}}) = F_h \cup F_s$ and $\mathit{Lbls}(\Phi_{F_{\text{ex}}}) = \{l_1, l_2, l_3, l_4\}$. The label-set $L = \{l_1\}$ is a minimal unsatisfiable core of $\Phi_{F_{\text{ex}}}$ as

$$\Phi_{F_{\text{ex}}}|_L = \{(x \vee z)^\emptyset, (\neg z)^\emptyset, (y \vee z)^\emptyset, (\neg x)^{\{l_1\}}\}$$

is unsatisfiable. L is also a minimal correction subset to $\Phi_{F_{\text{ex}}}$ as

$$\begin{aligned}\Phi_{F_{\text{ex}}}|_{\mathit{Lbls}(\Phi_{F_{\text{ex}}}) \setminus L} = \{&(x \vee z)^\emptyset, (\neg z)^\emptyset, (y \vee z)^\emptyset, (\neg y, \vee \neg z)^{\{l_2\}},\\ &(z \vee y)^{\{l_3\}}, (\neg z \vee y)^{\{l_4\}}\},\end{aligned}$$

is a LCNF formula satisfied by τ. As such τ is also an optimal solution to the LCNF-MaxSAT instance $\Phi_{F_{\text{ex}}}$. Converting $\Phi_{F_{\text{ex}}}$ back to a MaxSAT instance using the direct encoding results in the instance $F' = (F_h', F_s')$, where

$$\begin{aligned}F_h' = \{&(x \vee z), (\neg z), (y \vee z), (\neg x \vee a_1), (\neg y, \vee \neg z \vee a_2),\\ &(z \vee y \vee a_3), (\neg z \vee y \vee a_4)\} \text{ and}\\ F_s' = \{&(\neg a_1), (\neg a_2), (\neg a_3), (\neg a_4)\}.\end{aligned}$$

III. SAT Preprocessing for MaxSAT via LCNFs

Preprocessing has proven an integral part of the SAT-based approach to efficiently solving various types of real-world problem instances. However, to date there is only little work on the effects of preprocessing for MaxSAT, and the benefits of applying SAT-based preprocessing for MaxSAT still remain somewhat unclear. In fact, as shown in [22], many of the commonly used SAT preprocessing techniques, including bounded variable elimination (BVE) [20], self-subsuming resolution (SSR), or even subsumption elimination (SE), cannot be used directly on MaxSAT instances. As a remedy to this problem, in [22] liftings of VE, SSR, and SE to LCNF formulas were proposed. Essentially, the techniques can be applied on LCNFs by taking into account the natural restrictions implied by the SAT-level techniques on the label-sets of labelled clauses.

- *LCNF-lifting of the resolution rule:* The resolvent of two labelled clauses $(x \vee A)^{L_1}$ and $(\neg x \vee B)^{L_2}$ is $(x \vee A)^{L_1} \bowtie_x (\neg x \vee B)^{L_2} = (A \vee B)^{L_1 \cup L_2}$.
- *LCNF-lifting of BVE:* Let Φ_x and $\Phi_{\neg x}$, resp., denote the sets of labelled clauses C^L that contain the literal x and the literal $\neg x$, resp. The LCNF-level BVE allows for replacing $\Phi_x \cup \Phi_{\neg x}$ with $\Phi_x \bowtie_x \Phi_{\neg x} = \{A^{L_1} \bowtie_x B^{L_2} \mid A \in \Phi_x, B \in \Phi_{\neg x}, A \vee B \text{ not tautological}\}$ as long as the resulting formula does not contain more clauses than the original formula.
- *LCNF-lifting of SE:* A labelled clause A^{L_1} subsumes B^{L_2} if $A \subseteq B$ *and* $L_1 \subseteq L_2$. LCNF-level SE removes subsumed clauses until fixpoint.
- *LCNF-lifting of SSR:*
  Given labelled clauses $(l \vee A)^{L_1}$ and $(\neg l \vee B)^{L_2}$, if A^{L_1} subsumes B^{L_2}, replace $(\neg l \vee B)^{L_2}$ with B^{L_2}

Example 2: Eliminating z from $\Phi_{F_{\text{ex}}}$ gives the formula

$$\Phi^1_{F_{\text{ex}}} = \{(x)^{\emptyset}, (y)^{\emptyset}, (y)^{\{l_3\}}, (x \vee y)^{\{l_3\}}, (\neg y \vee x)^{\{l_2\}}, (x \vee y)^{\{l_4\}}, (y)^{\{l_4\}}, (y)^{\{l_3,l_4\}}, (\neg x)^{\{l_1\}}\}.$$

Applying labelled SE on $\Phi^1_{F_{\text{ex}}}$ results in the formula

$$\Phi^2_{F_{\text{ex}}} = \{(x)^{\emptyset}, (y)^{\emptyset}, (\neg x)^{\{l_1\}}\}.$$

Eliminating x from $\Phi^2_{F_{\text{ex}}}$ results in the formula

$$pre(\Phi_{F_{\text{ex}}}) = \{()^{\{l_1\}}, (y)^{\emptyset}\}.$$

The label-set $\{l_1\}$ is an LMCS of both $\Phi_{F_{\text{ex}}}$ and $pre(\Phi_{F_{\text{ex}}})$. Notice that the optimal costs of both formulas are the same.

Based on the fact that blocked clause elimination [27], [21] (BCE) does not affect the set of MUSes of any CNF formula [28], BCE is sound for MaxSAT. However, we note that in combination with LCNF-level variable elimination, self-subsuming resolution, and subsumption elimination, it is simpler to consider a straightforward lifting of blocked clause elimination[1] to LCNFs: a labelled clause C^L is *blocked* in an LCNF formula Φ if C is blocked in the CNF formula $Cl(\Phi)$. The soundness of BCE for LCNFs follows from the soundness of BCE for MaxSAT.

Here it is important to notice that, due to the resolution rule for LCNFs, bounded variable elimination can cause an increase in the size of the label-sets of the resulting labelled clauses. In particular, consider the encoding of MaxSAT as LCNF-MaxSAT. Even though each labelled clause corresponding to a soft clause in the original MaxSAT instance will have a singleton label-set, after LCNF-level preprocessing some of the clauses can have label-sets with more than one label. Direct encoding of the preprocessed weighted LCNF-MaxSAT instance as a MaxSAT instance will then add multiple new variables, corresponding to the labels of the labelled clauses, to the resulting soft clauses.

Integration of SAT-based preprocessing into the MaxSAT solving process is outlined in Figure 1, given a weighted partial MaxSAT instance $F = (F_h, F_s, c)$ as input.

For the solving (Step 3), any MaxSAT solver can be used by converting the LCNF back to standard MaxSAT.

[1]More generally, any *monotone* clause elimination procedure [28].

IV. Re-using Auxiliary Variables

Both the assumptions used in MaxSAT solving and the labels which enable SAT-based preprocessing require us to add a layer of new auxiliary variables to the working formula. In this work, we build on the idea of re-using the labels introduced in the SAT-based preprocessing step. We propose to identify variables in the input MaxSAT instance which could be re-used as labels *in the preprocessing step*.

A. Group Detecting Auxiliary Variables

Assume that we are given a weighted partial MaxSAT instance $F = (F_h, F_s, c)$ as input. We observe that MaxSAT instances may already contain variables that can be *directly re-used* as labels during preprocessing and as assumptions by a SAT solver. These variables can be easily identified from the instances by pattern matching. Especially, when viewing F as an LCNF-MaxSAT instance, this allows us to avoid introducing distinct labels for those clauses which contain variables that can be re-used as labels. In this work, we use the following simple scheme to identify such variables. Assume that F contains a literal[2] l that fulfills the following conditions.

(i) $(\neg l) \in F_s$.

(ii) $\neg l \notin C$ for any clause $C \in (F_h \cup F_s) \setminus \{(\neg l)\}$.

(iii) $l \notin C$ for any soft clause $C \in F_s$.

In words, we know that such a literal l only appears in the hard clauses of F and the literal $\neg l$ appears in a single soft unit clause in F. We note that such literals can be easily detected, and call this identification task *group detection*.

Example 3: Consider again the MaxSAT instance F_{ex} from Example 1. In this instance, the literal x satisfies the conditions of group detection and as such can be re-used as a label when converting F_{ex} to an LCNF-MaxSAT instance. This results in the formula

$$\Phi^g_{F_{\text{ex}}} = \{(z)^{\{x\}}, (\neg z)^{\emptyset}, (y \vee z)^{\emptyset}, (\neg y, \vee \neg z)^{\{l_1\}}, (z \vee y)^{\{l_2\}}, (\neg z \vee y)^{\{l_3\}}\}.$$

For concrete motivation for group detection, consider an arbitrary finite-domain constraint $\mathcal{C}$, and let $\mathsf{cnf}(\mathcal{C}) = \bigwedge_{i=1}^{k} C_i$

[2]Note here that a literal can be either a variable or its negation.

1) View $F = (F_h, F_s, c)$ as the LCNF-MaxSAT instance Φ_F as follows.
   - For each $C \in F_h$, introduce a labelled clause $C^{\emptyset}$.
   - For each $C \in F_s$, introduce a labelled clause $C^{\{l_C\}}$ where l_C is a distinct label for C with weight $c(C)$.
2) Apply the LCNF-liftings of BCE, VE, SSR, and SE on Φ_F to obtain the preprocessed LCNF $pre(\Phi_F)$.
3) Solve $pre(\Phi_F)$.

Fig. 1. Integrating SAT preprocessing into MaxSAT solving

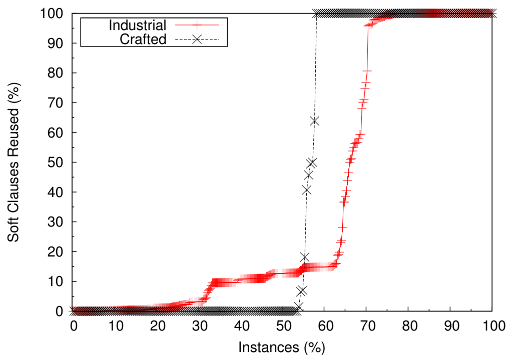

Fig. 2. Labels detected in the weighted partial benchmarks from MaxSAT Evaluation 2014

be a conjunctive normal form encoding of $\mathcal{C}$ (i.e., a representation of $\mathcal{C}$ as a set $\{C_1, \ldots, C_k\}$ of clauses). Now assume that $\mathcal{C}$ is a *soft* constraint, with an associated weight $W_{\mathcal{C}}$ defining the cost of not satisfying $\mathcal{C}$. On the level of Group MaxSAT, the soft constraint $\mathcal{C}$ with weight $W_{\mathcal{C}}$ can be represented as the soft group $\{C_1, \ldots, C_k\}$ with weight $W_{\mathcal{C}}$. For employing a standard MaxSAT solver, a natural way of encoding such a group-level MaxSAT representation [25] is to introduce an auxiliary variable $a_{\mathcal{C}}$, and to consider the set $\{(C_1 \lor a_{\mathcal{C}}), \ldots, (C_k \lor a_{\mathcal{C}})\}$ of *hard clauses* together with the soft clause $(\neg a_{\mathcal{C}})$ with weight $W_{\mathcal{C}}$.

While the proposed group detection procedure is simple, it can relatively often detect variables of interest in real MaxSAT benchmarks. Figure 2 shows the percentages of detected labels out of the number of soft clauses for each instance in the industrial and crafted weighted partial benchmarks from MaxSAT Evaluation 2014. The instances within each of the two categories are sorted by the percentage of detected labels. On a significant percentage of the instances, group detection was able to re-use all of the soft clauses in the input instance. Notice that this is only possible if all of the soft clauses in the input instances are unit, i.e., only contain a single literal.

B. *Group Detected Variables as Labels*

Group detection allows for re-using the detected "labelling" literals as labels when viewing F as an LCNF-MaxSAT instance. Concretely, we propose the computation steps outlined in Figure 3 as a refinement of the steps outlined in Figure 1. The essential difference here is that, instead of introducing a new label for each soft clause in order to apply SAT preprocessing (as in Figure 1), a new label is only introduced for soft clauses for which Step 0 identified no re-usable variables.

The soundness of group detection is formalized as follows.

Proposition 1: Let $F = (F_h, F_s, c)$ be a MaxSAT instance, Φ_F the LCNF-MaxSAT instance obtained from F following Step 1 in Figure 1, and Φ_F^g the LCNF-MaxSAT instance obtained from F following Steps 0-1 in Figure 3. The cost of the optimal solutions of Φ_F and Φ_F^g are the same.

Proof: (Sketch) We sketch the conversion of a solution τ of Φ_F^g to a solution of Φ_F. Let R be an LMCS such that τ satisfies $\Phi_F^g|_{Lbls(\Phi_F^g)\setminus R}$. Now construct an LMCS R' of Φ_F by including (i) each of the labels $l \in R \cap Lbls(\Phi_F)$, and (ii) for each label in $R \cap (Lbls(\Phi_F^g) \setminus Lbls(\Phi_F))$ (i.e., the group-detected labels), the label of the corresponding soft clause in Φ_F. It is easy to see that R and R' have the same cost. The fact that R' is an LMCS of Φ_F follows by considering the cases (i) and (ii) separately. The less obvious case (ii) follows from each such label in R being a pure literal in $\Phi_F|_{Lbls(\Phi_F)\setminus R'}$. ■

We end this section by noting that there is a connection between group detection and the preprocessing technique of labelled BVE. Consider again the MaxSAT instance F_{ex} from the previous examples. Only eliminating the variable x from $\Phi_{F_{\text{ex}}}$, the direct encoding of F_{ex} in LCNF results in the instance

$$\Phi_{F_{\text{ex}}} = \{(z)^{\{l_1\}}, (\neg z)^{\emptyset}, (y \lor z)^{\emptyset}, (\neg y, \lor \neg z)^{\{l_2\}}, \\ (z \lor y)^{\{l_3\}}, (\neg z \lor y)^{\{l_4\}}\}.$$

The same LCNF-MaxSAT instance $\Phi_{F_{\text{ex}}}^g$ (modulo label renaming) can also be obtained by encoding F_{ex} as LCNF with group detection.

V. MaxHS for Weighted LCNFs

In order to more thoroughly evaluate group detection, we make use of the LCNF-MaxHS algorithm developed in [19]. This allows us to test the impact of group detection without preprocessing, by re-using detected variables directly within a MaxSAT solver.

MaxHS ([23], [14], [15]) is a recent algorithm for weighted partial MaxSAT. It is a hybrid approach that alternates between

0. Apply group detection on $F = (F_h, F_s, c)$. Assume that group detection is able to identify a set $\mathcal{L}$ of labels associated with a subset $F_h' \subseteq F_h$ of hard clauses.

1'. Convert F into an LCNF-MaxSAT instance Φ_F^g as follows.
   - For each $C \in F_h'$, where $L \subseteq C$ for some subset $L \subseteq \mathcal{L}$, introduce the labelled clause $(C \setminus L)^L$. For each label $l \in L$, associate the weight $c((\neg l))$ with l.
   - For each $C \in F_h \setminus F_h'$, introduce the labelled clause $C^{\emptyset}$.
   - For each $C \in F_s$ that does not contain any literal in $\mathcal{L}$, introduce the labelled clause $C^{\{l_C\}}$, where l_C is a distinct label for C with weight $c(C)$.

2. Apply the labelled liftings of BCE, VE, SSR, and SE on Φ_F^g to obtain the preprocessed LCNF $pre(\Phi_F^g)$.
3. Solve $pre(\Phi_F^g)$.

Fig. 3. Combining group detection, SAT preprocessing, and LCNF-level MaxSAT solving

a SAT solver to compute unsatisfiable cores, and an integer programming (IP) solver to compute minimum-cost hitting sets (MCHS) over the found cores. In short, given a set of cores K for a formula F, MaxHS will invoke the IP solver to find a minimum-cost hitting set hs for K, and the SAT solver to solve the formula $F_h \cup (F_s \setminus hs)$. If the formula is unsatisfiable, a new core κ is derived and added to K and the process is repeated. Otherwise, hs implicitly hits *every* core of F with minimum cost, and the satisfying assignment to $F_h \cup (F_s \setminus hs)$ represents an optimal MaxSAT solution to F.

A. Lifting MaxHS

A lifting of the MaxHS algorithm to LCNFs, LCNF-MaxHS (Algorithm 1), was proposed in [19]. While LCNF-MaxHS closely follows the original MaxHS algorithm, it also makes a critical shift from the clause-centric view (with a single distinct auxiliary variable for each soft clause) to a label-centric view in which overlapping label-sets with more than one label are allowed. In more detail, LCNF-MaxHS maintains a set $\mathcal{L}$ of already identified cores (explicitly maintained on the LCNF-level as subsets of labels from $Lbls(\Phi)$) and a MCHS R for $\mathcal{L}$. During each iteration, an IP solver is used to find a MCHS R of $\mathcal{L}$. A SAT solver is then used to determine the satisfiability of $\Phi|_{Lbls(\Phi)\setminus R}$, the subformula of Φ induced by $Lbls(\Phi)\setminus R$.[3] If satisfiable, an optimal model was produced and the algorithm terminates. Otherwise, a new core L is obtained from the SAT solver, L is added to $\mathcal{L}$, and then the next iteration starts. This generalizes MaxHS to LCNFs while still maintaining correctness [19].

The motivation for LCNF-MaxHS in [19] was to allow for clean integration of SAT-based preprocessing for MaxSAT via LCNFs, and, importantly, to re-use the labels introduced in the preprocessing step directly auxiliary variables which can be used as assumptions within MaxHS. This avoids introducing an additional layer of variables in the SAT solver calls within MaxHS. This is implemented by the steps outlined in Figure 1. For Step 3, LCNF-MaxHS can be applied. In turn, LCNF-MaxHS can be realized by extending an implementation of the MaxHS algorithm to take as input the MaxSAT instance created by the *direct encoding* of $pre(\Phi_F)$. As proposed in [19], the SAT solver within MaxHS can be altered to work directly on the a_i variables as the assumptions, without having

[3]Notice that inducing a subformula by $Lbls(\Phi) \setminus R$ is analogous to removing all clauses present in the hitting set of the original MaxHS algorithm.

```
Input: An LCNF-MaxSAT instance $\Phi$
Output: An optimal solution $\tau$ for $\Phi$
$\mathcal{L} \leftarrow \emptyset$              // set of found unsat cores of $Lbls(\Phi)$
while true do
    $R \leftarrow$ MinCostHittingSet($\mathcal{L}$)
    (result, $L, \tau$) $\leftarrow$ SATSolve($\Phi|_{Lbls(\Phi)\setminus R}$)
    if result="satisfiable" then
        return $\tau$                         // solver returned SAT
    else
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{L\}$    // solver returned unsat core of
        $Lbls(\Phi)$
    end
end
```
Algorithm 1: LCNF-MaxHS, lifting of MaxHS to LCNFs

to introduce a new layer of auxiliary variables and without having to explicitly add the soft clauses $(\neg a_i)$ to the solver. Step 2 can be implemented using a SAT preprocessor on the direct encoding of Φ_F by restricting the preprocessor from removing any of the a_i variables corresponding to labels.

B. Understanding Group Detection with MaxHS

Eq-seeding was suggested in [14] to improve the effectiveness of the IP solver used in MaxHS. On the LCNF level, eq-seeding takes advantage of the fact that LCNF-MaxHS represents a unit labelled clause $(l)^{\{a_i\}}$ as the unit clause (l) augmented with an auxiliary variable (a_i), i.e., the clause $(l \vee a_i)$. In doing so, an implicit logical equivalence $l \leftrightarrow \neg a_i$ is created [14]. These equivalences need not be added to the SAT solver, but they can sometimes be used to derive linear constraints which can be added to the hitting set IP formulation of MaxHS. Let $C = (l_1 \vee \cdots \vee l_n)$ be a clause in F. A suitable linear constraint can be derived if for every $l_j \in C$, either l_j or $\neg l_j$ is equivalent to an auxiliary variable, or l_j is itself an auxiliary variable. Replacing each l_i with an equivalent auxiliary literal gives a linear at-least-one constraint equivalent to C, which can be added to the hitting set IP used in solving the minimum-cost hitting set problems encountered during search.

The next example demonstrates how group detection can improve the effectiveness of eq-seeding within LCNF-MaxHS. Encoding a MaxSAT instance F as an LCNF-MaxSAT instance Φ^g_F with group detection (via Steps 0–1 of Figure 3) can enable LCNF-MaxHS to derive more linear constraints during solving compared to the direct encoding (Step 1 in Figure 1) of F to Φ_F.

Example 4: Consider the (unweighted) MaxSAT instance $F' = (F'_h, F'_s)$ with

$$\begin{aligned} F'_h =& \{(g_1 \vee x_1), (g_1 \vee x_2), (g_2 \vee x_3), \\ & (g_2 \vee x_4), (\neg x_1 \vee \neg x_3)\} \text{ and} \\ F'_s =& \{(\neg g_1), (\neg g_2)\}. \end{aligned}$$

Converting F into an LCNF-MaxSAT instance $\Phi^g_{F'}$ using group detection results in the LCNF-MaxSAT instance

$$\Phi^g_{F'} = \{(x_1)^{\{g_1\}}, (x_2)^{\{g_1\}}, (x_3)^{\{g_2\}}, (x_4)^{\{g_2\}}, (\neg x_1 \vee \neg x_3)^{\emptyset}\}.$$

During solving, LCNF-MaxHS will treat these labelled clauses as the formula

$$\{(g_1 \vee x_1), (g_1 \vee x_2), (g_2 \vee x_3), (g_2 \vee x_4), (\neg x_1 \vee \neg x_3)\}$$

with auxiliary variables g_1 and g_2. Eq-seeding is able to infer the constraint $g_1 + g_2 \geq 1$ from the clause $(\neg x_1 \vee \neg x_3)$ and the equivalences $x_1 \leftrightarrow \neg g_1$ and $x_3 \leftrightarrow \neg g_2$. On the other hand, a direct encoding of F' in LCNF results in the instance

$$\begin{aligned} \{&(g_1 \vee x_1)^{\emptyset}, (g_1 \vee x_2)^{\emptyset}, (g_2 \vee x_3)^{\emptyset}, (g_2 \vee x_4)^{\emptyset}, \\ &(\neg x_1 \vee \neg x_3)^{\emptyset}, (\neg g_1)^{\{l_1\}}, (\neg g_2)^{\{l_2\}}\} \end{aligned}$$

which will be treated by LCNF-MaxHS as the formula

$$\begin{aligned} \{&(g_1 \vee x_1), (g_1 \vee x_2), (g_2 \vee x_3), (g_2 \vee x_4), \\ &(\neg x_1 \vee \neg x_3), (\neg g_1 \vee l_1), (\neg g_2 \vee l_2)\} \end{aligned}$$

with auxiliary variables l_1 and l_2. Here, eq-seeding will identify the equivalences $g_1 \leftrightarrow l_1$ and $g_2 \leftrightarrow l_2$ but cannot derive any linear constraints suitable for the hitting set IP.

Furthermore, the possibility of deriving more linear constraints can in some instances decrease the number of SAT and IP solver calls required by LCNF-MaxHS.

Example 5: Consider the LCNF-MaxSAT instances $\Phi_{F_{\text{ex}}}$ and $\Phi^g_{F_{\text{ex}}}$ from Examples 1 and 3, respectively. Notice that eq-seeding cannot derive any constraints from $\Phi_{F_{\text{ex}}}$. First we illustrate a possible (worst-case) execution of LCNF-MaxHS on $\Phi_{F_{\text{ex}}}$. Initially, LCNF-MaxHS invokes its SAT solver on the clauses of $\Phi_{F_{\text{ex}}}$. Assume that the SAT solver returns the core $L_1 = \{l_1, l_2\}$. At this point, the set of identified cores only contains L. Assume that the IP solver returns the minimum-cost hitting set $R = \{l_2\}$. Next, LCNF-MaxHS reiterates and invokes the SAT solver on the clauses of $\Phi_{F_{\text{ex}}}|_{Lbls(\Phi)\setminus\{l_2\}}$. The formula is still unsatisfiable. Assume that the SAT solver then returns the core $L_2 = \{l_1, l_3\}$. This time, the only minimum-cost hitting set over the set of all identified cores, $\{L_1, L_2\}$, is $\{l_1\}$. Finally, LCNF-MaxHS invokes the SAT solver on the clauses of $\Phi_F|_{Lbls(\Phi)\setminus\{l_1\}}$. This formula is satisfiable so the algorithm terminates and returns the satisfying assignment returned by the SAT solver. In total, two SAT- and IP-solver calls were needed. In contrast, as the constraint $x = 1$ can be derived from $\Phi^g_{F_{\text{ex}}}$ using eq-seeding, and every unsatisfiable core of $\Phi^g_{F_{\text{ex}}}$ has to include x, LCNF-MaxHS is guaranteed to require only a single SAT and IP solver call when solving $\Phi^g_{F_{\text{ex}}}$.

Group detection allows LCNF-MaxHS to derive more constraints using eq-seeding also in practice. Figure 4 shows a comparison between the number of constraints derivable by eq-seeding with and without group detection from the weighted partial benchmarks of the MaxSAT Evaluation 2014. While no further eq-seeding is obtained on the crafted instances, group detection improves eq-seeding on the industrial instances. A hypothetical explanation for the behavior on crafted instances is offered by the number of labels detected with group detection. As seen from Figure 2, on most crafted instances group detection detects either all soft clauses or no soft clauses. In fact, we observed a clear correlation between no soft clauses detected and no extra constraints derived by eq-seeding: whenever no soft clauses are detected, the LCNF-MaxSAT instances created with and without group detection are the same. For some intuition of the correlation between all soft clauses detected and no extra constraints derived, assume a MaxSAT instance F in which all soft clauses can be group detected. Let Φ^g_F and Φ_F the LCNF-MaxSAT instances created from F with and without group detection, respectively, and $\mathcal{C}$ a linear constraint derivable by eq-seeding from Φ^g_F but not from Φ_F. As discussed earlier, the fact that every soft clause was detected in F means that all soft clauses of F are unit soft clauses of form $(\neg l_i)$, where l_i is a label of Φ^g_F and $(\neg l_i)^{\{a_i\}}$ is a labelled clause of Φ_F. The fact that $\mathcal{C}$ was derivable from Φ^g_F but not from Φ_F suggests that there exists some set of labelled clauses of form $(x_i)^{\{l_i\}}$ and $(\bigvee_i x_i)^{\emptyset}$ in Φ^g_F. Then, the clauses $(x_i)^{\{l_i\}} \in \Phi^g_F$ correspond to clauses $(x_i \vee l_i)^{\emptyset} \in \Phi_F$, explaining why $\mathcal{C}$ cannot be derived from Φ. Such clauses, even though theoretically possible, would seem to only add unnecessary complexity to MaxSAT encodings arising from the real world. In such cases one could in the encoding substitute the (l_i)'s with the x_i variables in F.

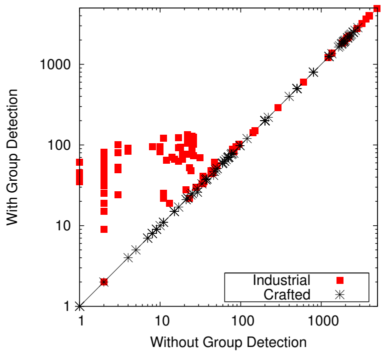

Fig. 4. Comparison of the number of linear constraints derivable by equivalence seeding with and without group detection on the weighted partial benchmarks from MaxSAT Evaluation 2014

VI. Experiments

We overview results from an empirical evaluation, with the aim of understanding the possible benefits of applying group detection in the MaxSAT solving process.

In the evaluation, we used the 410 instances from the weighted partial industrial category of MaxSAT Evaluation 2014 (http://www.maxsat.udl.cat/14/). The experiments were run on 2.53-GHz Intel Xeon quad-core machines with 32-GB RAM and Ubuntu Linux 12.04. A per-instance timeout of 1 h and memory limit of 30 GB were enforced. As the MaxSAT solvers, we used Eva [29], an award-winning core-based MaxSAT solver from the industrial weighted partial track of the 2014 MaxSAT Evaluation; and our own re-implementation of MaxHS that also enables the lifting of MaxHS to weighted LCNFs (Algorithm 1). This re-implementation includes the SAT solver tweaks and disjoint phase of [23], the non-optimal hitting set computations of [15], as well as the core minimization and eq-seeding techniques of [14]. MiniSAT 2.2.0 [30] is used as the underlying SAT solver, and IBM CPLEX 12.6.0 [31] is used to solve the minimum-cost hitting set IPs. For realizing LCNF-MaxHS, we extended our MaxHS implementation to take as input a list of variables suitable for use as assumptions. As the SAT preprocessor, we used Coprocessor 2.0 [32]. Its file parser was modified to automatically detect literals that can be re-used as labels, to add new auxiliary variables (labels) to the other soft clauses, and then using its whitelisting feature to forbid removal of all occurrences of every variable that represents a label during preprocessing.

We report on using the following preprocessing+solver combinations.

Eva: the Eva solver.

Eva-pre: Eva solver with preprocessing, using the direct encoding after preprocessing.

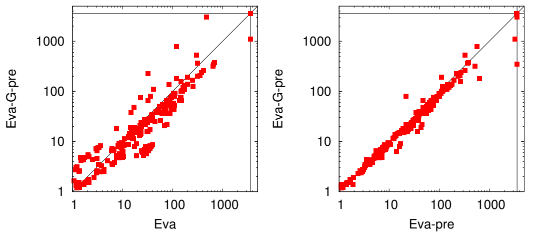

Fig. 5. Running times: Eva v Eva-G-pre (left); Eva-pre v Eva-G-pre (right).

Eva-G-pre: Eva solver, with group detection. Detected variables are re-used in preprocessing as labels. The direct encoding is used after preprocessing.

MHS2.5: MaxHS version 2.5 by the original MaxHS authors (http://www.maxhs.org/).

MHS: our re-implementation of MaxHS.

MHS-pre: MHS with preprocessing, using the direct encoding after preprocessing.

MHS-G-pre: MHS-pre, with group detection. Detected variables are re-used in preprocessing as labels. The direct encoding is used after preprocessing.

LMHS-G: MHS with group detection (i.e., LCNF-MaxHS with group detection). Detected variables are re-used as assumptions in the solver.

LMHS-pre: MHS with preprocessing, re-using auxiliary variables from preprocessing as assumptions in the solver.

LMHS-G-pre: MHS with group detection and preprocessing. Group detected variables are re-used as labels in preprocessing. Auxiliary variables from preprocessing are re-used as assumptions in the solver.

Result for Eva are shown in Figure 5. While no substantial differences in overall performance are observed, Eva-G-pre solves a majority of the instances faster than Eva, and timeouts on one less instance. Comparing Eva-G-pre to Eva-pre, i.e., looking at the additional effect of group detection in conjunction with preprocessing, we observe again that Eva-G-pre solves most instances slightly faster than Eva-pre, and Eva-pre timeouts on two more instances.

Results for variants of MHS are presented in Figures 6 and 7. We note that our MaxHS re-implementation is competitive with MaxHS 2.5. Second, we observe that employing SAT preprocessing in combination with plain MHS without re-using labels from the preprocessing step (MHS-pre) improves performance. Adding group detection (MHS-G-pre) improves further on MHS-pre. Perhaps the most interesting observation is that LMHS-G, i.e., applying group detection *solely* (i.e., without preprocessing) in conjunction with LCNF-MaxHS, is surprisingly effective; see also Figure 7. In more detail, first note that the MHS solver in Figure 6 could equivalently be viewed as LMHS without preprocessing or group detection, that is, Steps 1 and 3 of the computation outlined in Figure 1. As such, comparing the performance of MHS and LMHS-G gives an indication of the effect group detection alone has on MaxSAT solving. We see that group detection (LMHS-G) performs noticeably better than MHS, solving 16 instances more. One possible explanation for this performance—as discussed earlier—is eq-seeding; eq-seeding within LMHS-G was indeed able to derive more linear constraints than MHS on every single one of the instances that were solvable by LMHS-G but not MHS. Furthermore, on 92% of all benchmarks we observed that if LMHS-G solved the instance quicker than MHS, then it also needed fewer UNSAT cores. This further suggests that the reason for the difference in performance between MHS and LMHS-G is linked to the quality of the hitting sets returned by the IP solver. The connection between group detection and preprocessing is less clear. Comparing LMHS-G and LMHS-pre we surprisingly observe a similar level of performance, suggesting that group detection and preprocessing have similar effects on MaxSAT solving—although, a slight further performance increase is observed when combining the two (LMHS-G-pre).

VII. Conclusions

We proposed automatically detecting auxiliary variables from input MaxSAT instances that can be re-used in the SAT-based preprocessing step before MaxSAT solving. This further avoids adding unnecessary layers of auxiliary variables throughout the MaxSAT solving process both in preprocessing and also within the SAT-solver used in MaxSAT solvers. We empirically showed that such auxiliary variables can indeed be detected in real MaxSAT benchmarks, and that re-using these variables as assumptions gives—somewhat surprisingly—similar improvements on its own as applying SAT preprocessing on industrial weighted partial MaxSAT instances in terms of solving efficiency. As future work, we aim at studying ways of detecting and soundly exploiting auxiliary variables arising from clausal encodings of more complex soft constraints.

Acknowledgements

Work presented in this paper was funded by Academy of Finland, grants 251170 Centre of Excellence in Computational Inference Research, 276412, and 284591; Research Funds of the University of Helsinki; and the Emil Aaltonen Foundation.

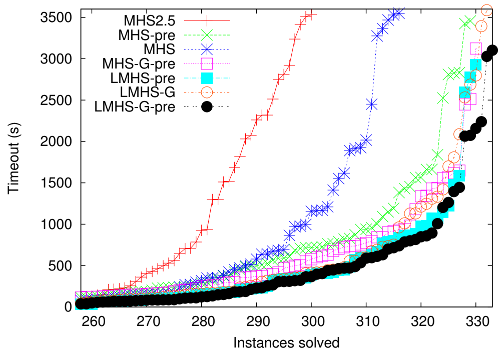

Fig. 6. Comparison of the different MaxHS variants

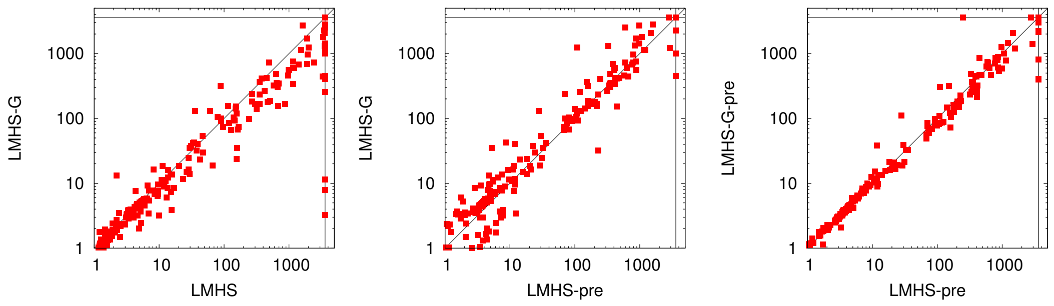

Fig. 7. Effect of group detection on runtimes: LMHS v LMHS-G (left); LMHS-G v LMHS-pre (middle); LMHS-pre v LMHS-G-Pre (right).

References

[1] C. Li and F. Manyà, "MaxSAT, hard and soft constraints," in *Handbook of Satisfiability*. IOS Press, 2009, pp. 613–631.

[2] A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva, "Iterative and core-guided MaxSAT solving: A survey and assessment," *Constraints*, vol. 18, no. 4, pp. 478–534, 2013.

[3] C. Ansótegui, M. Bonet, and J. Levy, "SAT-based MaxSAT algorithms," *Artificial Intelligence*, vol. 196, pp. 77–105, 2013.

[4] M. Jose and R. Majumdar, "Cause clue clauses: error localization using maximum satisfiability," in *Proc. PLDI*. ACM, 2011, pp. 437–446.

[5] C. Zhu, G. Weissenbacher, and S. Malik, "Post-silicon fault localisation using maximum satisfiability and backbones," in *Proc. FMCAD*. FMCAD Inc., 2011, pp. 63–66.

[6] J. Guerra and I. Lynce, "Reasoning over biological networks using maximum satisfiability," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 7514. Springer, 2012, pp. 941–956.

[7] J. Berg, M. Järvisalo, and B. Malone, "Learning optimal bounded treewidth bayesian networks via maximum satisfiability," in *Proc. AISTATS*, vol. 33. JMLR, 2014, pp. 86–95.

[8] K. Bunte, M. Järvisalo, J. Berg, P. Myllymäki, J. Peltonen, and S. Kaski, "Optimal neighborhood preserving visualization by maximum satisfiability," in *Proc. AAAI*. AAAI Press, 2014, pp. 1694–1700.

[9] J. Marques-Silva, M. Janota, A. Ignatiev, and A. Morgado, "Efficient model based diagnosis with maximum satisfiability," in *Proc. IJCAI*. AAAI Press, 2015.

[10] P. Saikko, B. Malone, and M. Järvisalo, "MaxSAT-based cutting planes for learning graphical models," in *Proc. CPAIOR*, ser. Lecture Notes in Computer Science, vol. 9075. Springer, 2015, pp. 345–354.

[11] J. Berg and M. Järvisalo, "Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability," *Artificial Intelligence*, 2015, in press.

[12] F. Heras, A. Morgado, and J. Marques-Silva, "Core-guided binary search algorithms for maximum satisfiability," in *Proc. AAAI*. AAAI Press, 2011.

[13] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "QMaxSAT: A partial Max-SAT solver," *Journal of Satisfiability, Boolean Modeling and Computation*, vol. 8, no. 1/2, pp. 95–100, 2012.

[14] J. Davies and F. Bacchus, "Exploiting the power of MIP solvers in MaxSAT," in *Proc. SAT*, ser. Lecture Notes in Computer Science, vol. 7962. Springer, 2013, pp. 166–181.

[15] ——, "Postponing optimization to speed up MAXSAT solving," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 8124. Springer, 2013, pp. 247–262.

[16] C. Ansótegui and J. Gabàs, "Solving (weighted) partial MaxSAT with ILP," in *Proc. CPAIOR*, ser. Lecture Notes in Computer Science, vol. 7874. Springer, 2013, pp. 403–409.

[17] A. Morgado, C. Dodaro, and J. Marques-Silva, "Core-guided MaxSAT with soft cardinality constraints," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 8656. Springer, 2014, pp. 564–573.

[18] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental cardinality constraints for MaxSAT," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 8656. Springer, 2014, pp. 531–548.

[19] J. Berg, P. Saikko, and M. Järvisalo, "Improving the effectiveness of SAT-based preprocessing for MaxSAT," in *Proc. IJCAI*. AAAI Press, 2015.

[20] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Proc. SAT*, ser. Lecture Notes in Computer Science, vol. 3569. Springer, 2005, pp. 61–75.

[21] M. Järvisalo, A. Biere, and M. Heule, "Blocked clause elimination," in *Proc. TACAS*, ser. Lecture Notes in Computer Science, vol. 6015. Springer, 2010, pp. 129–144.

[22] A. Belov, A. Morgado, and J. Marques-Silva, "SAT-based preprocessing for MaxSAT," in *Proc. LPAR-19*, ser. Lecture Notes in Computer Science, vol. 8312. Springer, 2013, pp. 96–111.

[23] J. Davies and F. Bacchus, "Solving MAXSAT by solving a sequence of simpler SAT instances," in *Proc. CP*, ser. Lecture Notes in Computer Science, vol. 6876. Springer, 2011, pp. 225–239.

[24] J. Argelich and F. Manyà, "Exact Max-SAT solvers for over-constrained problems," *Journal of Heuristics*, vol. 12, no. 4-5, pp. 375–392, 2006.

[25] F. Heras, A. Morgado, and J. Marques-Silva, "MaxSAT-based encodings for Group MaxSAT," *AI Communications*, vol. 28, no. 2, pp. 195–214, 2015.

[26] A. Belov and J. Marques-Silva, "Generalizing redundancy in propositional logic: Foundations and hitting sets duality," *CoRR*, vol. abs/1207.1257, 2012.

[27] O. Kullmann, "On a generalization of extended resolution," *Discrete Applied Mathematics*, vol. 96-97, pp. 149–176, 1999.

[28] A. Belov, M. Järvisalo, and J. Marques-Silva, "Formula preprocessing in MUS extraction," in *Proc. TACAS*, ser. Lecture Notes in Computer Science, vol. 7795. Springer, 2013, pp. 108–123.

[29] N. Narodytska and F. Bacchus, "Maximum satisfiability using core-guided MaxSAT resolution," in *Proc. AAAI*. AAAI Press, 2014, pp. 2717–2723.

[30] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proc. SAT*, ser. Lecture Notes in Computer Science, vol. 2919. Springer, 2003, pp. 502–518.

[31] IBM, "CPLEX Optimizer," 2015, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[32] N. Manthey, "Coprocessor 2.0 - A flexible CNF simplifier," in *Proc. SAT*, ser. Lecture Notes in Computer Science, vol. 7317. Springer, 2012, pp. 436–441.

Paper III

Jeremias Berg and Matti Järvisalo

Impact of SAT-Based Preprocessing on Core-Guided MaxSAT Solving

III

Impact of SAT-Based Preprocessing on Core-Guided MaxSAT Solving

Jeremias Berg( ) and Matti Järvisalo

Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki, Helsinki, Finland
jeremias.berg@cs.helsinki.fi

Abstract. We present a formal analysis of the impact of Boolean satisfiability (SAT) based preprocessing techniques on core-guided solvers for the constraint optimization paradigm of maximum satisfiability (MaxSAT). We analyze the behavior of two solver abstractions of the core-guided approaches. We show that SAT-based preprocessing has no effect on the best-case number of iterations required by the solvers. This implies that, with respect to best-case performance, the potential benefits of applying SAT-based preprocessing in conjunction with core-guided MaxSAT solvers are in principle solely a result of speeding up the individual SAT solver calls made during MaxSAT search. We also show that, in contrast to best-case performance, SAT-based preprocessing can improve the worst-case performance of core-guided approaches to MaxSAT.

1 Introduction

Real-world applications [1–18] of maximum satisfiability (MaxSAT) [19–21], the optimization counterpart of the famous Boolean satisfiability problem (SAT) [22,23], are increasing in numbers as recent breakthroughs in MaxSAT solvers [24–32] are making MaxSAT more and more competitive as a constraint optimization paradigm.

A great majority of state-of-the-art MaxSAT solvers for solving optimization problems from the real world are *core-guided* [20,21], heavily relying on the power of SAT solvers as very effective means of proving unsatisfiability of subsets of soft constraints, or *unsat cores*, in an iterative fashion towards an optimal solution. Thus new breakthroughs in techniques for speeding up SAT solvers also have the potential of directly speeding up MaxSAT solvers further. One particularly fruitful line of research on speeding up SAT solvers has been the development of effective preprocessing techniques [33–35], applied most typically before search, as well as most recently also as inprocessing [34], i.e., during SAT search. Compared to SAT, preprocessing for MaxSAT has seen some but arguably less progress so far [26,30,36–39]. Recently, ways of employing preprocessing techniques developed for pure SAT in the context of MaxSAT have

Work supported by Academy of Finland, grants 251170 COIN, 276412, 284591; and DoCS Doctoral School in Computer Science at the University of Helsinki.

M. Rueher (Ed.): CP 2016, LNCS 9892, pp. 66–85, 2016.
DOI: 10.1007/978-3-319-44953-1_5

been explored [26,30,40]. However, the impact of SAT-based preprocessing for MaxSAT solving seems to often be somewhat more modest than in the context of SAT solving [26,30,40]. The exact reasons for this difference are currently unclear; specifically, we are not aware of studies towards fundamental understanding on the potential of SAT-based preprocessing in the context of MaxSAT.

In this paper, we aim at providing further understanding on the potential of SAT-based preprocessing techniques in speeding up modern MaxSAT solvers. More specifically, we formally analyze the impact of SAT-based preprocessing techniques on the best-case and worst-case behavior of core-guided MaxSAT solvers [41–43]. As the basis of our analysis, we focus on two abstractions of MaxSAT solvers which together cover a number of modern core-guided MaxSAT solvers [25,30,42]. As the formal metric, we focus on the impact of SAT-based preprocessing on the best-case and worst-case number of iterations, which—although not the only possible metric—is a natural choice of metric applied in the literature for analyzing iterative SAT-based approaches in various problem settings [41–45] and which has also been subjected to some extent to empirical analysis for understanding specific MaxSAT solving approaches [46].

As the main contributions, considering *best-case performance* of the abstract core-guided solvers, we show that SAT-based preprocessing *has no effect* on the number of iterations required by the solvers. In fact, this is true regardless of assumptions on the type of cores (guaranteed-minimal or not) the underlying SAT solver (unsat core extractor) provides to the MaxSAT solvers; thus our analysis also sheds light on the impact of core minimization on the performance of the abstract core-guided solvers. Essentially, our results imply that, in terms of best-case performance—assuming optimal search heuristics—the potential benefits of applying SAT-based preprocessing in conjunction with core-guided MaxSAT solvers are solely a result of speeding up the individual SAT solver calls made during MaxSAT search. Furthermore, contrasting the results for best-case behavior, we also show that SAT-based preprocessing does, in cases, improve *worst-case* performance of core-guided MaxSAT solvers (without ever having a negative effect on the worst-case number of iterations).

This paper is organized as follows. After preliminaries on MaxSAT and SAT-based preprocessing for MaxSAT (Sect. 2), we detail abstractions of core-guided MaxSAT solvers we focus on (Sect. 3). Before detailed proofs of our results (provided in Sects. 5 and 6), we present a detailed overview of the main contributions (Sect. 4).

2 Preliminaries

Maximum satisfiability. For every Boolean variable x there are two literals: the positive literal x and the negative literal $\neg x$. A clause C is a disjunction of literals, and a CNF formula F is a conjunction of clauses. When convenient, we treat a clause as a set of literals and a CNF formula as a set of clauses. We denote by $\textsc{Var}(F)$ the set of variables appearing in F. A truth assignment is a function $\tau\colon \textsc{Var}(F) \to \{0,1\}$. A clause C is satisfied by τ if $\tau(l) = 1$ for a positive literal or $\tau(l) = 0$ for a negative literal $l \in C$. A CNF formula F is satisfied by τ if τ satisfies all clauses $C \in F$. A formula F is satisfiable if there is a truth assignment that satisfies it, otherwise it is unsatisfiable.

A (weighted partial) MaxSAT instance $F = (F_h, F_s, w)$ consists of two CNF formulas, F_h (hard clauses) and F_s (soft clauses), together with a function $w \colon F_s \to \mathbb{N}$ assigning a positive weight $w(C)$ to each $C \in F_s$. If $w(C) = 1$ for all $C \in F_s$, the instance is unweighted. An (unsatisfiable) *core* of a MaxSAT instance F is a subset $\kappa \subseteq F_s$ such that $\kappa \wedge F_h$ is unsatisfiable. A core is minimal (a MUS) if no $\kappa_s \subset \kappa$ is a core of F. We denote the set of all MUSes of F by $\mathsf{mus}(F)$. For a subset $S \subseteq F_s$ and clause $C \in S$, C is *necessary* for S if $F_h \wedge S$ is unsatisfiable and $F_h \wedge (S \setminus \{C\})$ is satisfiable.

An assignment τ that satisfies F_h is a *solution* to a MaxSAT instance F. For a solution τ, let $\textsc{cost}(F, \tau) = \sum_{C \in F_s} w(C) \cdot (1 – \tau(C))$, i.e., the sum of the weights of soft clauses in F not satisfied by τ. A solution τ is optimal if $\textsc{cost}(F, \tau) \leq \textsc{cost}(F, \tau')$ for every solution τ'; we denote the cost of F, i.e., the value $\textsc{cost}(F, \tau)$ for optimal solutions τ, by $\textsc{cost}(F)$. Given a MaxSAT instance F, the MaxSAT problem asks to find an optimal solution to F.

SAT-Based Preprocessing for MaxSAT. Preprocessing is today an integral part of SAT solving [33,34]. Consisting of applying a combination of satisfiability-preserving simplification (or rewriting) rules on the input CNF formula F to obtain a preprocessed CNF formula $\mathsf{pre}(F)$, a central aim of preprocessing is to speed up the runtime of a SAT solver so that the combined preprocessing time and solving time on $\mathsf{pre}(F)$ is shorter than the runtime of the solver on F. Several preprocessing techniques for SAT have been proposed. In this work we will focus on bounded variable elimination, subsumption elimination, self-subsuming resolution, and blocked clause elimination, as perhaps the most common preprocessing techniques in modern SAT solving.

Resolution. Given two clauses $C = C_1 \vee l$ and $D = D_1 \vee \neg l$ of F, the *resolution rule* states that the clause $C \bowtie_l D = C_1 \vee D_1$, called the *resolvent*, can be inferred by *resolving* on the literal l. This is lifted to two sets $S_l \subseteq F$ and $S_{\neg l} \subseteq F$ of clauses that contain the literal l and $\neg l$, respectively, by $S_l \bowtie_l S_{\neg l} = \{C \bowtie_l D \mid C \in S_l, D \in S_{\neg l},$ and $C \bowtie_l D$ is not a tautology$\}$.

Bounded Variable Elimination (BVE) [33]. For a variable $x \in \textsc{Var}(F)$, denote by F_x ($F_{\neg x}$) the clauses of F containing the literal x ($\neg x$). If $|F_x \bowtie_x F_{\neg x}| \leq |F_x \cup F_{\neg x}|$, the BVE rule allows converting the formula F to $(F \setminus (F_x \cup F_{\neg x})) \cup (F_x \bowtie_x F_{\neg x})$.

Subsumption Elimination (SE). A clause $C \in F$ subsumes another clause $D \in F$ if $C \subseteq D$. The SE rule allows for removing subsumed clauses from F.

Self-Subsuming Resolution (SSR). Given two clauses $C, D \in F$ s.t. $C = C_1 \vee l$, $D = D_1 \vee \neg l$ for a literal l and $D_1 \subseteq C_1$, the SSR rule allows for replacing C by C_1.

Blocked Clause Elimination (BCE) [47]. A clause $C \in F$ is blocked if it contains a literal $l \in C$ s.t $C \bowtie_l D$ is a tautology for all $D \in F_{\neg l}$. BCE allows removing blocked clauses from F.

Example 1. Consider the CNF formula
$F = \{(x \vee y), (\neg t \vee \neg z), (\neg z \vee y), (\neg y \vee z), (z \vee t), (x), (y \vee t), (z \vee t \vee x)\}$. Due to the clause (x), SE allows for removing $(x \vee y)$ and $(z \vee t \vee x)$. After this, using BVE to eliminate z, results in the formula $\mathsf{pre}(F) = \{(\neg t \vee \neg y), (t \vee y), (x)\}$.

As shown in [26], many important SAT preprocessing techniques, including BVE, SE, and SSR, cannot be used directly on MaxSAT instances. However, a correct lifting on these techniques for MaxSAT is enabled by the so-called *labelled CNF* (LCNF) framework [26,48]. The LCNF framework enables correct applications of SAT-based preprocessing techniques on a MaxSAT instance $F = (F_h, F_s, w)$ using the procedure outlined in Fig. 1. Each soft clause $C \in F_s$ is augmented with a fresh *label variable* l_C (Step 1). Then SAT preprocessing is applied on the CNF formula $F_h \cup F_s^a$ (Step 2). To ensure correctness in terms of MaxSAT, the preprocessor needs to be restricted from resolving on any of the label variables. The hard clauses of $\mathsf{pre}(F)$ are the clauses output by the SAT preprocessor on $F_h \cup F_s^a$ (Step 3). The soft clauses of $\mathsf{pre}(F)$ contain a unit negation of each label variable that has not been eliminated by preprocessing; the weight function w^P assigns to each $(\neg l_C)$ the same weight as was assigned to C by w (Step 4). Finally, the procedure returns the preprocessed instance $\mathsf{pre}(F) = (\mathsf{pre}(F)_h, \mathsf{pre}(F)_s, w^P)$ (Step 5). The soft clauses of $\mathsf{pre}(F)$ are all unit soft clauses $(\neg l_C)$ where the variable l_C was added to some soft clause $C \in F_s$ of the original instance F in Step 1. Due to BVE, the variable l_C might appear in more than one hard clause of $\mathsf{pre}(F)$ and there might be literals that have been eliminated entirely from the formula during preprocessing.

1. $F_s^a = \{(C \vee l_C) \mid C \in F_s,\ l_C$ is a fresh variable$\}$.
2. Run VE, SSR, SE, and BCE on $F_h \cup F_s^a$ until fixpoint to obtain pre $(F)_h$.
3. pre $(F)_s = \{(\neg l_C) \mid \exists C' \in$ pre $(F)_h, l_C \in C'\}$.
4. $w^P((\neg l_C)) = w(C)$ for all $(\neg l_C) \in$ pre $(F)_s$.
5. Return pre $(F) =$ (pre $(F)_h$, pre $(F)_s, w^P)$.

Fig. 1. Applying SAT-based preprocessing to MaxSAT instance $F = (F_h, F_s, w)$.

Example 2. Let $F = (F_h, F_s)$ be an unweighted MaxSAT instance with
$F_h = \{(x \vee y), (z), (z \vee t)\}$ and $F_s = \{(\neg x), (\neg y), (\neg t)\}$. Augmenting the soft clauses with the label variables l_1, l_2, and l_3 to form $F_s^a = \{(\neg x \vee l_1), (\neg y \vee l_2), (\neg t \vee l_3)\}$, and applying SAT-based preprocessing (BVE and SE) results in the instance $\mathsf{pre}(F)$ with $\mathsf{pre}(F)_h = \{(l_1 \vee l_2), (z)\}$ and $\mathsf{pre}(F)_s = \{(\neg l_1), (\neg l_2)\}$. Notice that preprocessing eliminates the label l_3.

Correctness of SAT-based preprocessing for MaxSAT is summarized as follows [26].

Theorem 1 **([26]).** *Let F be a MaxSAT instance and $\mathsf{pre}(F)$ the instance resulting from preprocessing F according to the procedure in Fig. 1. The following* *hold: (i)* $\text{COST}(F) = \text{COST}(\mathit{pre}(F))$*; (ii) any optimal solution to* $\mathit{pre}(F)$ *restricted to* $\text{VAR}(F)$ *is an optimal solution to* F*; and (iii)* $\{C_1, \ldots, C_n\} \in \mathit{mus}(F)$ *iff* $\{(\neg l_{C_1}), \ldots, (\neg l_{C_n})\} \in \mathit{mus}(\mathit{pre}(F))$.

3 Core-Guided MaxSAT Algorithms

In this section we detail the two abstractions of MaxSAT algorithms we analyze in this work: *CG* and *HS*. Both are examples of so-called *core-guided MaxSAT solvers*, one of the most successful current MaxSAT solving approaches with several variants, e.g. [28,31,42,49–52]. *CG* (Fig. 2 left) is the same abstraction as studied in [53]. CG works by iteratively calling a SAT solver to extract unsatisfiable cores and ruling out each of the found cores by exploiting *cardinality constraints*. HS (Fig. 2 right) follows the implicit hitting set approach to MaxSAT [54,55], iteratively using a SAT solver to extract unsatisfiable cores, and an exact minimum-cost hitting set algorithm to compute hitting sets over the found cores.

In more detail, at each iteration i, CG checks the satisfiability of a working formula F_w^i, which initially contains all clauses in the input formula, using a SAT solver. If F_w^i is satisfiable, CG returns the satisfying assignment τ returned by the SAT solver restricted onto the variables of F. Otherwise, the SAT solver returns a core κ^i of F_w^i. Finally, CG forms the next working formula F_w^{i+1} by processing the core κ^i. The exact method in which CG processes κ^i is left abstract. Following [53], we consider algorithms that extend soft clauses with blocking variables and impose hard linear (in)equalities over the blocking variables. More precisely, CG is allowed to modify the soft clauses $C \in F_s^i$ by two operations: **Relax**(C) and **Clone**(C, w).

```
CG:
$F_w^1 \leftarrow F_h \cup F_s$
for $i$=1... do
    $(\mathit{result}, \kappa, \tau) \leftarrow \text{SATSOLVE}(F_w^i)$
    if $\mathit{result}$="satisfiable" then
        return $\tau$            // optimal solution
    else
        // SAT solver returned unsat core
        $F_w^i = (F_w^i \setminus \kappa)$
        $F_w^{i+1} \leftarrow \text{PROCESS}(F_w^i, \kappa)$
    end
end
```

```
HS:
$\mathcal{K} \leftarrow \emptyset$       // set of found unsat cores of $F$
$F_w \leftarrow (F_h \cup F_s)$
while true do
    $H \leftarrow \text{MINCOSTHITTINGSET}(\mathcal{K})$
    $F_w \leftarrow F_h \cup (F_s \setminus H)$
    $(\mathit{result}, \kappa, \tau) \leftarrow \text{SATSOLVE}(F_w)$
    if $\mathit{result}$="satisfiable" then
        return $\tau$            // optimal solution
    else
        // SAT solver returned unsat core
        $\mathcal{K} \leftarrow \mathcal{K} \cup \{\kappa\}$
    end
end
```

Fig. 2. Abstractions of MaxSAT solvers: CG (left) and HS (right), given a MaxSAT instance $F = (F_h, F_s, w)$ as input.

– **Relax**(C) allows replacing C by $C \vee b$ where b is a new *blocking variable* not appearing anywhere else in the formula.
– **Clone**(C, w) allows adding a soft duplicate C' of C to the formula and relaxing C' by calling **Relax**(C'). The (relaxed) *clone* C' is assigned weight w, and w is subtracted from the weight of C (C is discarded once it has weight 0).

In addition to these operations, CG is also allowed to add hard linear (in)equalities (cardinality, or more precisely, pseudo-Boolean, constraints) over the blocking variables. Given a cardinality constraint $\sum w_i \cdot x_i \circ K$ over variables x_i, constants w_i, and $\circ \in \{=, <, \leq\}$, we denote by $\text{CNF}(\sum w_i \cdot x_i \circ K)$ a CNF encoding of such a constraint. Following most core-guided MaxSAT algorithm implementations, we place two important restrictions on how CG can process the cores it encounters. First, the cardinality constraints are not allowed to mention any of the variables in the initial formula F. Second, if the algorithm extracts n cores during solving an instance F, and w^i_m is the smallest weight over all clauses in the ith core extracted, the optimum cost of F is $\textsc{cost}(F) = \sum_{i=1}^{n} w^i_m$. A concrete example of an algorithm fitting the CG model is the WPM1 algorithm [50], concurrently proposed as WMSU1 [51], as an extension of the classical Fu-Malik algorithm [49] to weighted MaxSAT. Given a core κ^i, WPM1 first computes w^i_m. Then it calls **Clone**(C^i, w^i_m) for each $C^i \in \kappa^i$ and adds an *exactly-one* constraint over the blocking variables added during the cloning operation.

HS is a hybrid algorithm, instantiated in [25,55], that uses a SAT solver for core extraction from a working formula F_w, initially all clauses of the working formula. Given a collection $\mathcal{K}$ of extracted cores, HS uses an exact algorithm (an integer programming solver in practice) to find a minimum-cost hitting set hs over $\mathcal{K}$. The working formula is then updated to contain all clauses of F except for the soft clauses in hs, and the SAT solver invoked again. If the working formula is satisfiable, the satisfying assignment obtained is an optimal solution to F. Otherwise another core is obtained and the search continues with hitting set computation.

4 Overview of Results

In this section we give an overview of the main contributions of this paper. The algorithm-dependent formal proofs are provided after this overview in Sects. 5 and 6.

We start by first defining the metric with respect to which we perform the formal analysis. The definition, intuitively matching with the number of iterations made by the abstract MaxSAT solvers considered, relies on the concept of *core traces*. Informally, a core trace T is a finite sequence of MaxSAT cores matching a possible execution of a core-guided MaxSAT solver. More formally, given a MaxSAT instance F and $\mathcal{A} \in \{\text{CG}, \text{HS}\}$, a sequence $(\kappa^1, \ldots, \kappa^n)$ of cores is an $\mathcal{A}$ *core trace* on F if there exists an execution of $\mathcal{A}$ on F such that (i) the core extracted by $\mathcal{A}$ at iteration i is κ^i; and (ii) $\mathcal{A}$ terminates after having encountered all cores in the sequence (i.e., the $(n+1)$th SAT solver call is satisfiable). For a core trace T, we denote by $|T|$ the number of cores in T, i.e., the *length* of T. Whenever appropriate, we refer to $\mathcal{A}$ core traces on F simply as $\mathcal{A}$ traces on F.

As the metric under analysis, we consider both the *minimum* and *maximum length* over all possible $\mathcal{A}$ traces for different choices of $\mathcal{A}$. More specifically, for $\mathcal{A} \in \{\text{CG}, \text{HS}\}$, we analyze the relative minimum and maximum lengths of core traces for the following variants of $\mathcal{A}$.

- $\mathcal{A}_{\mathsf{pre}}$: $\mathcal{A}$ applied after SAT-based preprocessing (recall Fig. 1).
- $\mathcal{A}^{\mathsf{mus}}$: $\mathcal{A}$ using a SAT solver that is guaranteed to return a MUS when invoked on an unsatisfiable formula (notice that an $\mathcal{A}^{\mathsf{mus}}$ trace contains only MUSes).
- $\mathcal{A}^{\mathsf{mus}}_{\mathsf{pre}}$: $\mathcal{A}^{\mathsf{mus}}$ applied after SAT-based preprocessing.

For a MaxSAT instance F, we denote by $\mathsf{minlen}(\mathcal{A}, F)$ and $\mathsf{maxlen}(\mathcal{A}, F)$ the minimum and maximum length $\mathcal{A}$ traces on F, respectively, or in other words, the best-case and worst-case number of iterations required by $\mathcal{A}$ for solving F.

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\boxed{\mathcal{A}} \arrow[r, bend left, "\text{Corollary 2, Corollary 5}"] \arrow[d, bend left, "\text{Observation 1}"] & \boxed{\mathcal{A}_{\text{pre}}} \arrow[l, bend left, "\text{Corollary 2, Corollary 5}"] \arrow[d, bend left, "\text{Observation 1}"] \\
\boxed{\mathcal{A}^{\text{mus}}} \arrow[u, bend left, "\text{Proposition 2, Proposition 6}"] \arrow[r, bend left, "\text{Observation 2, Proposition 5}"] & \boxed{\mathcal{A}^{\text{mus}}_{\text{pre}}} \arrow[u, bend left, "\text{Proposition 2, Proposition 6}"] \arrow[l, bend left, "\text{Observation 2, Proposition 5}"]
\end{tikzcd}$$

Fig. 3. Best-case performance in the number of iterations of $\mathcal{A} \in \{\text{CG}, \text{HS}\}$. Here $X \rightarrow Y$ iff $\mathsf{minlen}(X, F) \leq \mathsf{minlen}(Y, F)$ for all instances F.

Results. We provide a full characterization of the effect of preprocessing on the maximum and minimum length of core traces on F. The results on the best-case performance (minimum lengths of core traces) are summarized in Fig. 3 for $\mathcal{A} \in \{\text{CG}, \text{HS}\}$. In the figure, an edge $X \rightarrow Y$ indicates that, for any MaxSAT instance F, the shortest X core trace on F is at most as long as the shortest Y core trace on F. Analogously, our results for the worst-case performance (maximum lengths of core traces) are summarized in Fig. 4. Here the edge $X \rightarrow Y$ indicates that, for any MaxSAT instance F, the longest X core trace on F is at most as long as the longest Y core trace on F; $X \nrightarrow Y$ indicates that $X \rightarrow Y$ does not hold. In words, we will provide in the following sections detailed proofs for the fact that SAT-based preprocessing cannot lower the minimum number of iterations required by CG or HS. For some intuition, we will show that for $\mathcal{A} \in \{\text{CG}, \text{HS}\}$, one of the shortest $\mathcal{A}$ core traces on any MaxSAT instance F is also a $\mathcal{A}^{\mathsf{mus}}$ trace, and that preprocessing cannot alter the MUS structure nor the $\mathcal{A}^{\mathsf{mus}}$ traces on F.

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\boxed{\mathcal{A}} \arrow[r, bend left, "\text{Proposition 3, Proposition 8}", "/" marking] \arrow[d, bend left, "\text{Proposition 3, Proposition 9}", "/" marking] & \boxed{\mathcal{A}_{\text{pre}}} \arrow[l, bend left, "\text{Observation 3, Proposition 7}"] \arrow[d, bend left, "\text{Proposition 4, Proposition 9}", "/" marking] \\
\boxed{\mathcal{A}^{\mathsf{mus}}} \arrow[u, bend left, "\text{Observation 1}"] \arrow[r, bend left, "\text{Observation 2 Proposition 5}"] & \boxed{\mathcal{A}^{\mathsf{mus}}_{\text{pre}}} \arrow[u, bend left, "\text{Observation 1}"] \arrow[l, bend left, "\text{Observation 2 Proposition 5}"]
\end{tikzcd}$$

Fig. 4. Worst-case performance in the number of iterations of $\mathcal{A} \in \{\text{CG}, \text{HS}\}$. Here $X \rightarrow Y$ iff $\mathsf{maxlen}(X, F) \leq \mathsf{maxlen}(Y, F)$ for all F, and $X \nrightarrow Y$ indicates that $X \rightarrow Y$ does not hold.

In contrast, we will also show that preprocessing can improve the worst-case performance of both of the algorithms. Intuitively, this is due to the fact that preprocessing can remove soft clauses that are not members of any MUSes of F and hence do not contribute to the unsatisfiability of F, but still might force either algorithm to iterate unnecessarily many times.

We proceed now throughout Sects. 5 and 6 by providing formal proofs for all of the results summarized in Figs. 3 and 4. Before the more involved proofs, we start with an algorithm-independent observation and an auxiliary result that makes the remaining proofs simpler by allowing us to assume MaxSAT instances to have a specific form without loss of generality.

Observation 1. *For $\mathcal{A} \in \{CG, HS\}$ and any MaxSAT instance F, any $\mathcal{A}^{\mathsf{mus}}$ trace on F is also an $\mathcal{A}$ trace on F. Hence $\mathsf{maxlen}(\mathcal{A}^{\mathsf{mus}}, F) \leq \mathsf{maxlen}(\mathcal{A}, F)$ and $\mathsf{minlen}(\mathcal{A}^{\mathsf{mus}}, F) \geq \mathsf{minlen}(\mathcal{A}, F)$.*

Finally, in the remaining proofs, we will use the fact that Theorem 1 guarantees that SAT-based preprocessing does not affect the set of MUSes of F in terms of of the mapping $(\neg l_C) \rightarrow C$ between the soft clauses of $\mathsf{pre}(F)$ and F. In order to avoid explicitly referring to this mapping in every proof, we will employ a technical observation from [40]. More specifically, *we will assume for the remaining part of this paper* that the soft clauses $C \in F_s$ of each MaxSAT instance F have already been augmented with label variables l_C to form the hard clause $C \vee l_C$ and the soft clause $(\neg l_C)$. In other words, we will assume that all soft clauses of F are unit negative literals $(\neg l_C)$ with the variable l_C not appearing negatively in any other clause and only appearing positively among the hard clauses. Under this assumption, the literals appearing in the soft clauses of F can be reused as label variables while preprocessing [40], thus removing the need of adding any new variables. Hence $\mathsf{pre}(F)_s \subseteq F_s$, and Theorem 1 can be simplified.

Corollary 1 (of Theorem 1). *Let F be a MaxSAT instance and $\mathsf{pre}(F)$ the instance resulting after preprocessing F. Then $\mathsf{mus}(F) = \mathsf{mus}(\mathsf{pre}(F))$.*

Most importantly, our assumption on the form of MaxSAT instances *does not affect core traces.* A proof for this auxiliary result is provided in Appendix A.

Proposition 1. *Let $F = (F_h, F_s, w)$ be a MaxSAT instance, and $F^P = (F_h \cup F_s^a, F_s^P, w^P)$ the MaxSAT instance with $F_s^a = \{C \vee l_C \mid C \in F_s,\ l_C$ is a fresh variable$\}$, $F_s^P = \{(\neg l_C) \mid C \in F_s\}$, and $w^P((\neg l_C)) = w(C)$. The following observations hold.*

1. *$\textsc{cost}(F) = \textsc{cost}(F^P)$, and the optimal solutions of F are the same as the optimal solutions of F^P restricted to $\textsc{Var}(F)$.*
2. *For $\mathcal{A} \in \{HS, CG\}$, there is a one-to-one mapping between the $\mathcal{A}$ core traces on F and F^P of equal length.*

5 Impact of Preprocessing on HS

We continue with formal proofs of our main results for HS. An essential intuition for these proofs is that HS only extracts cores of the original instance. In other words, an HS core trace on any F only contains cores of the original instance F.

We first analyze best-case performance. The first observation shows that preprocessing does not affect the lengths of HS MUS traces in a significant way.

Observation 2.
For any MaxSAT instance F, $\mathsf{minlen}(HS^{\mathsf{mus}}, F) = \mathsf{minlen}(HS^{\mathsf{mus}}_{\mathsf{pre}}, F)$.

Proof. (Sketch) By Corollary 1 we obtain $\kappa \in \mathsf{mus}(F)$ iff $\kappa \in \mathsf{mus}(\mathsf{pre}(F))$. The fact that an $\mathrm{HS}^{\mathsf{mus}}$ trace on F only contains MUSes of F implies that T is an $\mathrm{HS}^{\mathsf{mus}}$ trace on F iff it is an $\mathrm{HS}^{\mathsf{mus}}_{\mathsf{pre}}$ trace on F. □

Next we show that executions of $\mathrm{HS}^{\mathsf{mus}}$ are always shortest executions of HS.

Proposition 2.
For any MaxSAT instance F, $\mathsf{minlen}(HS, F) \geq \mathsf{minlen}(HS^{\mathsf{mus}}, F)$ and $\mathsf{minlen}(HS_{\mathsf{pre}}, F) \geq \mathsf{minlen}(HS^{\mathsf{mus}}_{\mathsf{pre}}, F)$.

Proof. We will show that $\mathsf{minlen}(\mathrm{HS}, F) \geq \mathsf{minlen}(\mathrm{HS}^{\mathsf{mus}}, F)$ for any F, and thus $\mathsf{minlen}(\mathrm{HS}_{\mathsf{pre}}, F) \geq \mathsf{minlen}(\mathrm{HS}^{\mathsf{mus}}_{\mathsf{pre}}, F)$ as well. Let $T = (\kappa^1, \ldots, \kappa^n)$ be an arbitrary HS core trace on F. Let hs^* be a minimum-cost hitting set over $\{\kappa^1, \ldots, \kappa^n\}$ for which $F \setminus hs^*$ is satisfiable. The statement follows by constructing an $\mathrm{HS}^{\mathsf{mus}}$ trace T_m on F s.t. $|T_m| \leq |T|$. As each $\kappa^i \in T$ is a core of F, all contain at least one MUS $m \subseteq \kappa^i$. Consider the set $\mathcal{M}$ of at most n MUSes of F constructed as follows. (1) Let $\mathcal{M}^1 = \{m^1\}$, where m^1 is any MUS contained in κ^1; (2) let $\mathcal{M}^i = \mathcal{M}^{i-1} \cup \{m^i\}$, where $m^i \subseteq \kappa^i$ is a MUS such that $m^i \notin \mathcal{M}^{i-1}$ if any exist, else let $\mathcal{M}^i = \mathcal{M}^{i-1}$. We obtain $\mathcal{M}^n = \mathcal{M}$ of size $|\mathcal{M}| = k \leq n$ such that each $m \in \mathcal{M}$ is a subset of some $\kappa^i \in T$.

We show that $\mathcal{M}$ can be ordered to form an HS^{mus} trace on F of length at most k, since if a minimum-cost hitting set hs over any proper subset $\mathcal{M}_s \subset \mathcal{M}$ hits all $m \in \mathcal{M}$, then hs^* is also a minimum-cost hitting set over $\mathcal{M}_s$, and HS^{mus} can terminate. As $F \setminus hs^*$ is satisfiable, hs^* is also a hitting set over $\mathcal{M}$ and over $\mathcal{M}_s$. Furthermore, as each $m \in \mathcal{M}$ is a subset of some $\kappa^i \in T$ and each $\kappa^i \in T$ contains a MUS in $\mathcal{M}$, hs^* is a minimum-cost hitting set of $\mathcal{M}$. Finally, as hs is a hitting set over $\mathcal{M}$ the cost of hs is not less than the cost of hs^*. Hence hs^* is a minimum-cost hitting set of $\mathcal{M}_s$, so the hitting set computation could have returned hs^*, thus allowing HS^{mus} to terminate. □

A simple corollary is that shortest executions of HS and HS_{pre} are of equal length.

Corollary 2. *For any MaxSAT instance F, $\mathsf{minlen}(HS, F) = \mathsf{minlen}(HS_{\mathsf{pre}}, F)$.*

Proof.
Observation 1 and Proposition 2 establish $\mathsf{minlen}(\text{HS}, F) = \mathsf{minlen}(\text{HS}^{\mathsf{mus}}, F)$ and $\mathsf{minlen}(\text{HS}_{\mathsf{pre}}, F) = \mathsf{minlen}(\text{HS}^{\mathsf{mus}}_{\mathsf{pre}}, F)$. Together with Observation 2 this implies $\mathsf{minlen}(\text{HS}, F) = \mathsf{minlen}(\text{HS}^{\mathsf{mus}}, F) = \mathsf{minlen}(\text{HS}^{\mathsf{mus}}_{\mathsf{pre}}, F) = \mathsf{minlen}(\text{HS}_{\mathsf{pre}}, F)$. □

We move on to the worst-case results. Corollary 1 can be used to show that valid executions of HS_{pre} are also valid executions of HS on any MaxSAT instance.

Observation 3.
For any MaxSAT instance F, $\mathsf{maxlen}(HS_{\mathsf{pre}}, F) \leq \mathsf{maxlen}(HS, F)$.

Proof. As $\mathsf{pre}(F)_s \subseteq F_s$ and any MUS of $\mathsf{pre}(F)$ is a MUS of F, any core of $\mathsf{pre}(F)$ is a core of F. □

Finally for this section, we prove the three $X \nrightarrow Y$ edges in Fig. 4 for HS. For this, we need as a witness a family of MaxSAT instances $F(n)$ and a X core trace T on $F(n)$ s.t. $|T| > \mathsf{maxlen}(Y, F(n))$.

Proposition 3.
There is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t. $\mathsf{maxlen}(HS, F(n)) \geq n$ and $\mathsf{maxlen}(HS^{\mathsf{mus}}, F(n)) = \mathsf{maxlen}(HS_{\mathsf{pre}}, F(n)) = 1$.

Proof. Fix n and let $F(n)_h = \{(x \vee y)\} \cup \{(x \vee y \vee z_i) \mid i = 1, \ldots, n\}$ and $F(n)_s = \{(\neg x), (\neg y)\} \cup \{(\neg z_i) \mid i = 1, \ldots, n\}$ with $w((\neg x)) = w((\neg y)) = n$ and $w((\neg z_i)) = 1$ for all i. Now $\textsc{cost}(F(n)) = n$ and $\mathsf{mus}(F(n)) = \{\{(\neg x), (\neg y)\}\}$, explaining why $\mathsf{maxlen}(\text{HS}^{\mathsf{mus}}, F(n)) = 1$. A linear-length HS core trace on $F(n)$ is $(\kappa^1, \ldots, \kappa^n)$, where $\kappa^i = \{(\neg x), (\neg y), (\neg z_i)\}$. HS cannot terminate before extracting all n cores. To see this, consider an earlier iteration $i < n$. The weight of the hitting set $\{(\neg z_j) \mid j = 1, \ldots, i\}$ over $\mathcal{K}^i = \{\kappa^1, \ldots, \kappa^i\}$ is $i < n = w((\neg x)) = w((\neg y))$ and as such any minimum-cost hitting set over $\mathcal{K}^i$ can not contain $(\neg x)$ or $(\neg y)$, preventing HS from terminating. Hence $\mathsf{maxlen}(\text{HS}, F(n)) \geq n$.

However, due to the clause $(x \vee y)$, SE allows the removal of the clause $(x \vee y \vee z_i)$ for all i. Hence $\mathsf{pre}(F(n))$ has $\mathsf{pre}(F(n))_h = \{(x \vee y)\}$ and $\mathsf{pre}(F(n))_s = \{(\neg x), (\neg y)\}$. The only core of $\mathsf{pre}(F(n))$ is $\{(\neg x), (\neg y)\}$, and thus $\mathsf{maxlen}(\text{HS}_{\mathsf{pre}}, F(n)) = 1$. $\square$

Proposition 4. *For any n, there is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t. $\mathsf{maxlen}(HS_{\mathsf{pre}}, F(n)) \geq n$ and $\mathsf{maxlen}(HS^{\mathsf{mus}}_{\mathsf{pre}}, F) = 1$.*

Proof. Fix n and let

$$F(n)_h = \{(x_{1,2} \vee x_{1,3} \vee \neg x_{2,3}), (E \vee x_{2,3})\} \cup \tag{1}$$

$$\bigcup_{i=4}^{n+3} \{(x_{1,2} \vee x_{2,i} \vee \neg x_{1,i}), (x_{1,i} \vee x_{1,3} \vee \neg x_{3,i}), (x_{3,i} \vee x_{2,i} \vee \neg x_{2,3})\} \cup \tag{2}$$

$$\{(x_{T,x} \vee x_{x,y} \vee \neg x_{T,y}), (x_{T,x} \vee x_{T,y} \vee \neg x_{x,y}) \mid 1 \leq x, y \leq n+3\} \tag{3}$$

and $F(n)_s = \{(\neg x_{1,2}), (\neg x_{1,3}), (\neg E)\} \cup \{(\neg x_{2,i}) \mid i = 4, \ldots, n+3\}$ with $w((\neg x_{1,2})) = w((\neg x_{1,3})) = w((\neg E)) = n$ and $w((\neg x_{2,i})) = 1$ for all i. The hard clauses on row 3 are included in order to prevent preprocessing from simplifying $F(n)$ in any way. Intuitively, $F(n)$ encodes hard transitivity constraints over an undirected graph with each node having degree at least 4. Hence $\mathsf{pre}(F(n)) = F(n)$ at it suffices to show $\mathsf{maxlen}(\text{HS}, F(n)) \geq n$ and $\mathsf{maxlen}(\text{HS}^{\mathsf{mus}}, F) = 1$. Both arguments are similar to Proposition 3. As $\mathsf{mus}(F(n)) = \{\{(\neg x_{1,2}), (\neg x_{1,3}), (\neg E)\}\}$, it follows that $\mathsf{maxlen}(\text{HS}^{\mathsf{mus}}, F) = 1$. A linear-length HS core trace on $F(n)$ is $(\kappa^1, \ldots, \kappa^n)$, where $\kappa^i = \{(\neg x_{1,2}), (\neg x_{1,3}), (\neg E), (\neg x_{2,i+3})\}$. $\square$

6 Impact of Preprocessing on CG

We start the analysis for CG by linking CG core traces with optimum cost.

Observation 4. *Let $T = (\kappa^1, \ldots, \kappa^n)$ be a CG or CG$^{\mathsf{mus}}$ core trace on a MaxSAT instance F, and $w^i = \min\{w(C^i) \mid C^i \in \kappa^i\}$. The cost of F is $\textsc{cost}(F) = \sum_{i=1}^n w^i$.*

An important corollary of Observation 4 is that no proper subsequence of a CG or CG$^{\mathsf{mus}}$ core trace on F can in itself be a CG or CG$^{\mathsf{mus}}$ trace on F.

The proofs on CG, in contrast to HS, need to consider the fact that the ith core κ^i in a CG core trace on F is not a core of F, but rather, of the working formula F^i instead. Following this, a relationship between the cores of F^i and the cores of F was derived in [53]. After necessary definitions and restatement of the result of [53], we will prove an analogous result regarding the relationship between the MUSes of F^i and F, which proves useful for obtaining our main results for CG.

6.1 Cores and MUSes of Working Formulas of CG

We follow here definitions from [53]. Let F be a MaxSAT instance and F^i the working formula of CG on iteration i when invoked on F. Let $\mathbf{card}^i$ be the set of all cardinality constraints added to F by CG during iterations $1, \ldots, i$. Thus the hard clauses of F^i are $F_h^i = F_h \cup \mathbf{card}^i$. We denote by $\mathbf{soln}(\mathbf{card}^i)$ the set of truth assignments satisfying $\mathbf{card}^i$ and not assigning any of the variables in F. Given any $\tau\colon \textsc{Var}(F) \to \{0,1\}$ and $\alpha \in \mathbf{soln}(\mathbf{card}^i)$, $(\tau{:}\alpha)$ is the truth assignment over the variables of F^i that assigns all variables of F according to τ and the rest according to α; $(\tau{:}\alpha)$ is well-defined as the auxiliary cardinality constraints are not allowed to mention variables in F. For any $\beta \in \mathbf{soln}(\mathbf{card}^i)$ and $S^i \subseteq F_s^i$, the *reduction* of S^i wrt β, $S^i|_\beta$ is obtained by (1) removing from S^i all clauses satisfied by β; (2) removing from each remaining clause $C^i \in S^i$ all blocking variables, i.e., all literals falsified by β; and (3) setting the weights of each $C^i \in S^i$ back to their original weights in F (removing duplicates). The restriction $\mathrm{R}(C^i) \in F_s$ of a soft clause $C^i \in F_s^i$ is obtained by (1) removing all added blocking variables from C^i; (2) removing all clones of C^i from the instance; and (3) setting the weight of C^i back to its original weight in F. Restriction is lifted to a set $S^i \subseteq F_s^i$ by $\mathrm{R}(S^i) = \{\mathrm{R}(C^i) \mid C^i \in S^i\}$. Notice that $S^i|_\beta \subseteq \mathrm{R}(S^i) \subseteq F_s$. With these definitions we can now restate a central result from [53].

Theorem 2 (Adapted from [53]).
A set $\kappa^i \subseteq F_s^i$ is a core of F^i iff $\kappa^i|_\beta$ is a core of F for all $\beta \in \mathbf{soln}(\mathbf{card}^i)$.

We will now prove an analogous characterization of the MUSes of F^i.

Theorem 3. *A set $M^i \subseteq F_s^i$ is a MUS of F^i iff there is a collection $\Upsilon \subseteq$ $\mathsf{mus}(F)$ s.t.*

1. $\mathrm{R}(M^i) = \bigcup_{M \in \Upsilon} M$;
2. *for each $M \in \Upsilon$, there is an $\alpha \in \mathbf{soln}(\mathbf{card}^i)$ s.t. $M \subseteq M^i|_\alpha$ and $M' \not\subseteq M^i|_\alpha$ for all other $M' \in \Upsilon$; and*
3. *for each $\alpha \in \mathbf{soln}(\mathbf{card}^i)$, there is an $M \in \Upsilon$ s.t. $M \subseteq M^i|_\alpha$.*

Note that condition 3 is equivalent to the requirement of Theorem 2 for the set M^i being a core of F^i, since $M^i|_\alpha \subseteq \mathrm{R}(M^i)$ and $M^i|_\alpha$ should be unsatisfiable for all α.

Before proving Theorem 3, consider the following example for more intuition.

Example 3. Consider the unweighted MaxSAT instance $F = (F_h, F_s)$ with $F_h = \{(x_1 \vee x_2 \vee x_3), (x_3 \vee x_4 \vee x_5), (x_5 \vee x_6 \vee x_7), (x_8)\}$ and $F_s = \cup_{i=1}^{8}\{(\neg x_i)\}$. Invoke WPM1 [50] on F and assume that it first processes the core $\{(\neg x_3), (\neg x_4), (\neg x_5)\}$. Afterwards the working formula F^2 is $F_h^2 = F_h \cup \{\mathrm{CNF}(r_1 + r_2 + r_3 = 1)\}$ and $F_s^2 = \{(\neg x_1), (\neg x_2), (\neg x_3 \vee r_1), (\neg x_4 \vee r_2), (\neg x_5 \vee r_3), (\neg x_6), (\neg x_7)(\neg x_8)\}$. Now $\mathbf{card}^2 = \{\mathrm{CNF}(r_1 + r_2 + r_3 =$

$1)\}$ and the set $\mathbf{soln}(\mathbf{card}^2)$ contains three assignments α^i, $i = 1, \ldots, 3$, assigning r_i to 1 and the others to 0. By Theorem 2, the set $\kappa^2 = \{(\neg x_1), (\neg x_2), (\neg x_3 \vee r_1), (\neg x_5 \vee r_3), (\neg x_6), (\neg x_7)\}$ is a core of F^2 as each $\kappa^2|_{\alpha^i}$ is a core of F. For example, $\kappa^2|_{\alpha^1} = \{(\neg x_1), (\neg x_2), (\neg x_5), (\neg x_6), (\neg x_7)\}$. In order to use Theorem 3 to show that κ^2 is also a MUS of F^2, note that $\mathrm{R}(\kappa^2) = \{(\neg x_1), (\neg x_2), (\neg x_3), (\neg x_5), (\neg x_6), (\neg x_7)\} = \{(\neg x_1), (\neg x_2), (\neg x_3)\} \cup \{(\neg x_5), (\neg x_6), (\neg x_7)\}$, where $\{(\neg x_1), (\neg x_2), (\neg x_3)\}$ and $\{(\neg x_5), (\neg x_6), (\neg x_7)\}$ are MUSes of F. Condition 2 of Theorem 3 follows since the only MUS in $\kappa^2|_{\alpha^3}$ is $\{(\neg x_1), (\neg x_2), (\neg x_3)\}$ and the only MUS in $\kappa^2|_{\alpha^1}$ is $\{(\neg x_5), (\neg x_6), (\neg x_7)\}$.

Next we prove Theorem 3. We begin by some lemmas. Assume for each of them that CG is invoked on an instance F and that F^i is the working formula on iteration i.

Lemma 1. *Let M^i be a MUS of F^i and $C^i \in M^i$. There is an $\alpha \in \mathbf{soln}(\mathbf{card^i})$ s.t. $\mathrm{R}(C^i)$ is necessary for $M^i|_\alpha$.*

Proof. By Theorem 2, $M^i|_{\alpha'}$ is a core of F for all $\alpha' \in \mathbf{soln}(\mathbf{card}^i)$. Hence it suffices to show that $M^i|_\alpha \setminus \mathrm{R}(C^i)$ is not a core for some α. Consider the assignment $(\tau{:}\alpha)$ satisfying $F_h^i \wedge (M^i \setminus \{C^i\})$, guaranteed to exist as M^i is a MUS of F^i. Now τ satisfies $F_h \wedge (M^i \setminus \{C^i\})|_\alpha = F_h \wedge (M^i|_\alpha \setminus \mathrm{R}(C^i))$ as required. □

Corollary 3. *For any MUS M^i of F^i, $\mathrm{R}(M^i) \subseteq \bigcup \mathsf{mus}(F)$.*

Corollary 4. *For any MUS M^i of F^i, there is an irreducible $\Upsilon \subseteq \mathsf{mus}(F)$ s.t. $\mathrm{R}(M^i) = \bigcup_{M \in \Upsilon} M$.*

Proof. Take Υ as the smallest collection of MUSes of F for which $\mathrm{R}(M^i) \subseteq \bigcup_{M \in \Upsilon} M$; by Corollary 3 such a collection exists. We claim that $\bigcup_{M \in \Upsilon} M \subseteq \mathrm{R}(M^i)$, from which irreducibility follows directly by minimality of Υ. Fix an arbitrary $C_e \in M$ in some $M \in \Upsilon$. By minimality of Υ, there is a clause $C^i \in M^i$ for which the only MUS of Υ containing $\mathrm{R}(C^i)$ is M. By Lemma 1, there exists a β for which $\mathrm{R}(C^i)$ is necessary for $M^i|_\beta$. As $M^i|_\beta \subseteq \mathrm{R}(M^i) \subseteq \bigcup_{M \in \Upsilon} M$ and the only MUS in Υ containing $\mathrm{R}(C^i)$ is M, we have $C_e \in M \subseteq M^i|_\beta \subseteq \mathrm{R}(M^i)$, establishing $C_e \in \mathrm{R}(M^i)$ and $\bigcup_{M \in \Upsilon} M \subseteq \mathrm{R}(M^i)$. □

We are now ready to prove Theorem 3.

Proof (of Theorem 3). A collection $\Upsilon \subseteq \mathsf{mus}(F)$ satisfying condition 1 exists by Corollary 4. For condition 2, we use the fact that the set Υ is irreducible. Let $M \in \Upsilon$ be arbitrary. Similarly to the proof of Corollary 4, we can find a $C^i \in M^i \in \Upsilon$ and $\alpha \in \mathbf{soln}(\mathbf{card}^i)$ s.t $\mathrm{R}(C^i) \notin M'$ for any other $M' \in \Upsilon$ and $\mathrm{R}(C^i)$ is necessary for $M^i|_\alpha$, implying that the only MUS in $M^i|_\alpha$ is M. Finally, condition 3 follows from M^i being a core of F^i and Theorem 2.

What remains is to show that subset $M^i \subseteq F_s^i$ satisfying conditions 1–3 is a MUS of F^i. By condition 3 and Theorem 2, M^i is a core of F^i. Hence we only need to show that it is minimally unsatisfiable, i.e., $F_h^i \wedge (M^i \setminus \{C^i\})$ is satisfiable for all $C^i \in M^i$. Fix $C^i \in M^i$ and let Υ be the collection of MUSes of

F for which $\mathrm{R}(M^i) = \bigcup_{M \in \Upsilon} M$. Consider any MUS $M_C \in \Upsilon$ s.t. $\mathrm{R}(C^i) \in M_C$. By condition 2, there is an $\alpha \in \mathbf{soln}(\mathbf{card}^i)$ for which the only MUS (of F) in $M^i|_\alpha \subseteq \mathrm{R}(M^i)$ is M_C. For such α, $F_h \wedge M^i|_\alpha \setminus \{\mathrm{R}(C_i)\}$ is satisfied by some τ. Hence $(\tau{:}\alpha)$ satisfies $F_h \wedge \mathbf{card}^i \wedge (M^i \setminus \{C^i\}) = F_h^i \wedge (M^i \setminus \{C^i\})$. $\square$

Finally, we note that each condition in Theorem 3 is necessary.

Example 4. Consider again the MaxSAT instance F from Example 3. The set $\{(\neg x_1), (\neg x_2), (\neg x_3 \vee r_1)\}$ is an example of a non-MUS of F^1 satisfying conditions 1–2 and the set $\{(\neg x_1), (\neg x_2), (\neg x_3 \vee r_1), (\neg x_5 \vee r_3), (\neg x_6), (\neg x_7), (\neg x_8)\}$ is an example of a non-MUS of F^1 satisfying conditions 1 and 3.

6.2 Results on Core Trace Lengths

We proceed with proofs on the number of iterations for CG. With respect to best-case, preprocessing does not affect the lengths of $\mathrm{CG}^{\mathsf{mus}}$ traces significantly.

Proposition 5.
For any MaxSAT instance F, $\mathsf{minlen}(CG^{\mathsf{mus}}, F) = \mathsf{minlen}(CG^{\mathsf{mus}}_{\mathsf{pre}}, F)$.

Proof. We show that a $T_m = (m^1, \ldots, m^n)$ is a $\mathrm{CG}^{\mathsf{mus}}$ trace on F iff it is a $\mathrm{CG}^{\mathsf{mus}}_{\mathsf{pre}}$ trace on F. We prove the left-to-right direction, the other is similar. We will show that there is an execution of $\mathrm{CG}^{\mathsf{mus}}_{\mathsf{pre}}$ on F for which the ith MUS extracted is m^i and which terminates only after extracting all MUSes of T_m. The termination follows from no proper subset of a $\mathrm{CG}^{\mathsf{mus}}$ trace being a core trace in itself.

We show that each m^i is a MUS of $\mathsf{pre}(F)^i$ by induction. By Corollary 1, m^1 is a MUS of $\mathsf{pre}(F)$. Assume that $\mathrm{CG}^{\mathsf{mus}}$ has extracted and processed the MUSes $(m^1, \ldots, m^{i-1})$ from $\mathsf{pre}(F)$ and consider the ith iteration. As m^i is a MUS of F^i, by Theorem 3 there is an $\Upsilon \subseteq \mathsf{mus}(F)$ s.t. $\mathrm{R}(m^i) = \cup_{m \in \Upsilon} m$. For $m^i \in \mathsf{mus}(\mathsf{pre}(F)^i)$, we show that Υ satisfies the conditions of Theorem 3 in $\mathsf{pre}(F)$ as well. By Corollary 1, each $m \in \Upsilon$ is a MUS of $\mathsf{pre}(F)$. For the other two conditions, note that by induction, the set of cardinality constraints $\mathbf{card}_p^i$ added to $\mathsf{pre}(F)$ after processing the MUSes $m^1, \ldots, m^{i-1}$ is the same as the set $\mathbf{card}^i$ added to F after processing the same sequence of MUSes. Hence $\alpha \in \mathbf{soln}(\mathbf{card}_p^i)$ iff $\alpha \in \mathbf{soln}(\mathbf{card}^i)$, which implies the two other conditions of Theorem 3. $\square$

Next we show that some shortest execution of CG is also an execution of $\mathrm{CG}^{\mathsf{mus}}$.

Proposition 6.
For any MaxSAT instance F, $\mathsf{minlen}(CG^{\mathsf{mus}}, F) \leq \mathsf{minlen}(CG, F)$ and $\mathsf{minlen}(CG^{\mathsf{mus}}_{\mathsf{pre}}, F) \leq \mathsf{minlen}(CG_{\mathsf{pre}}, F)$.

Proof. (Sketch) We prove $\mathsf{minlen}(\mathrm{CG}^{\mathsf{mus}}, F) \leq \mathsf{minlen}(\mathrm{CG}, F)$; the same proof works for $\mathsf{minlen}(\mathrm{CG}^{\mathsf{mus}}_{\mathsf{pre}}, F) \leq \mathsf{minlen}(\mathrm{CG}_{\mathsf{pre}}, F)$ as well. Let $T = (\kappa^1, \ldots, \kappa^n)$ be a CG trace on F. We construct a $\mathrm{CG}^{\mathsf{mus}}$ trace $T_m = (m^1, \ldots, m^k)$ on F of at most the same length recursively. For intuition, on each iteration i $\text{CG}^{\textsf{mus}}$ processes a subset of the clauses CG would have processed on the ith iteration of the execution corresponding to T. Hence, if $\textbf{card}^i_m$ and $\textbf{card}^i$ are the set of cardinality constraints added to F by the ith iteration on the execution corresponding to T_m and T, respectively, then any $\alpha \in \textbf{soln}(\textbf{card}^i_m)$ can be extended to a solution to $\textbf{card}^i$ by assigning the remaining variables to 0.

Let m^1 be an MUS of F contained in κ^1. Assume that $\text{CG}^{\textsf{mus}}$ has extracted the MUSes m^j for $j = 1, \ldots, i-1$ s.t each $m^j \subseteq \kappa^j$. Consider the ith iteration and the current working formula F^i_m. As κ^i is a core of F^i, the ith working formula on the execution corresponding to T, by Theorem 2 $\kappa^i|_\beta$ is a core of F for all $\beta \in \textbf{soln}(\textbf{card}^i)$. Hence $\kappa^i|_\beta$ is also a core of F for all $\beta \in \textbf{card}^i_m$. Applying Theorem 2 gives that κ^i is a core of F^i_m. Hence it also contains a MUS m^i of F^i_m. For termination of $\text{CG}^{\textsf{mus}}$, note that $\min_{C^i \in \kappa^i}\{w(C^i)\} \leq \min_{C^i \in m^i}\{w(C^i)\}$ for every i. Since $\sum_{i=1}^n \min_{C^i \in \kappa^i}\{w(C^i)\} = \textsc{cost}(F)$, termination of $\text{CG}^{\textsf{mus}}$ occurs at the latest after n iterations on the execution corresponding T_m. □

Finally, we show that the shortest executions of CG and $\text{CG}_{\textsf{pre}}$ are of the same length.

Corollary 5. *For any MaxSAT instance F, $\textsf{minlen}(CG, F) = \textsf{minlen}(CG_{\textsf{pre}}, F)$.*

Proof. Proposition 6 and Observation 1 imply $\textsf{minlen}(\text{CG}, F) = \textsf{minlen}(\text{CG}^{\textsf{mus}}, F)$ and $\textsf{minlen}(\text{CG}^{\textsf{mus}}_{\textsf{pre}}, F) = \textsf{minlen}(\text{CG}_{\textsf{pre}}, F)$. Together with Proposition 5 we obtain $\textsf{minlen}(\text{CG}, F) = \textsf{minlen}(\text{CG}^{\textsf{mus}}, F) = \textsf{minlen}(\text{CG}^{\textsf{mus}}_{\textsf{pre}}, F) = \textsf{minlen}(\text{CG}_{\textsf{pre}}, F)$. □

We move on to worst-case results for CG. We begin by showing that valid executions of $\text{CG}_{\textsf{pre}}$ are also valid executions of CG.

Proposition 7.
For any MaxSAT instance F, $\textsf{maxlen}(CG, F) \geq \textsf{maxlen}(CG_{\textsf{pre}}, F)$.

Proof. We show that a $\text{CG}_{\textsf{pre}}$ trace $T = (\kappa^1, \ldots, \kappa^n)$ on F is also a CG trace on F. The termination of CG only after n iterations follows from the cost-preserving properties of preprocessing and Observation 4. We show that each κ^i is a valid core of F^i by induction. The case $i = 1$ follows from $\textsf{pre}(F)_s \subseteq F_s$ and Corollary 1. Assume next that all κ^j for $j < i$ have been cores of F^j and consider κ^i. By Theorem 2, $\kappa^i|_\beta$ is a core of $\textsf{pre}(F)$ for all $\beta \in \textbf{soln}(\textbf{card}^i_p)$, where $\textbf{card}^i_p$ is the set of cardinality constraints added to $\textsf{pre}(F)$ after processing cores $\kappa^1, \ldots, \kappa^{i-1}$. By induction, this set is exactly the same as set of cardinality constraints $\textbf{card}^i$ added to F after processing the same cores. As any core of $\textsf{pre}(F)$ is a core of F, it follows that $\kappa^i|_\beta$ is a core of F for all $\beta \in \textbf{soln}(\textbf{card}^i)$. We conclude that κ^i is a core of F^i. □

Finally, two families of instances witness the $\nrightarrow$ edges in Fig. 4 for CG.

Proposition 8.
There is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t. $\textsf{maxlen}(CG, F(n)) \geq n$ and $\textsf{maxlen}(CG^{\textsf{mus}}, F(n)) = \textsf{maxlen}(CG_{\textsf{pre}}, F(n)) = 1$.

Proof. (Sketch) Consider again the instance $F(n)$ constructed in the proof of Proposition 3. We showed that $\mathsf{maxlen}(\mathrm{HS}^{\mathsf{mus}}, F) = \mathsf{maxlen}(\mathrm{HS}_{\mathsf{pre}}, F) = 1$. This also holds for CG. A linear-length CG core trace $(\kappa^1, \ldots, \kappa^n)$, on F can be constructed iteratively as follows: $\kappa^1 = \{(\neg x), (\neg y), (\neg z_1)\}$ and $\kappa^i = \{(\neg x)^c_{i-1}, (\neg y)^c_{i-1}, (\neg z_i)\}$ where $(\neg x)^c_{i-1}$ and $(\neg y)^c_{i-1}$ are duplicates of the original clauses added on iteration $i - 1$. The existence of such duplicates for all n iterations follows from $w((\neg x)) = w((\neg y)) = n$ and $w((\neg z_i)) = 1$. The termination of CG after the nth iteration follows from Observation 4 as the smallest weight among the clauses in each κ^i is 1. □

Proposition 9. *There is a family of MaxSAT instances $F(n)$ with $\mathcal{O}(n)$ soft clauses s.t. $\mathsf{maxlen}(CG_{\mathsf{pre}}, F(n)) \geq n$ and $\mathsf{maxlen}(CG^{\mathsf{mus}}_{\mathsf{pre}}, F) = 1$.*

Proof. (Sketch) $F(n)$ is the same as for HS and the proof follows Proposition 4. A linear-length CG core trace can be constructed similarly to Proposition 8 by replacing clauses in the linear-length HS trace from Proposition 4 with duplicates of original clauses where required. □

7 Conclusions

We formally analyzed the effect of SAT-based preprocessing, as well as core minimization, on the performance of core-guided MaxSAT solvers. As a main result, we showed that SAT-based preprocessing has no effect on the best-case number of iterations required by the solvers but can improve on the worst-case. In terms of best-case performance, the potential benefits of applying SAT-based preprocessing in conjunction with core-guided MaxSAT solvers are thus in principle—assuming optimal search heuristics—solely in speeding up individual SAT solver calls made during MaxSAT search. Simultaneously, our analysis also revealed an analogous result on the impact of core minimization in core-guided MaxSAT solvers. Our results motivate further work on developing MaxSAT-specific preprocessing techniques capable of affecting the MaxSAT algorithms on a more general level. In contrast, SAT-based preprocessing does in cases have a positive effect on the worst-case number of iterations. Of independent interest, we established a formal characterization of how the underlying MUS structure is altered by iterative revisions performed by CG solvers on MaxSAT instances (Theorem 3), thus sharpening the main results of [53].

Appendix

A Proof of Proposition 1

(1) If an optimal solution τ to F assigns $\tau(C) = 0$, then an optimal solution τ^P to F_P has to assign $F_P(l_C) = 1$. Similarly, if $\tau(C) = 1$, then τ^P can assign $\tau^P(l_C) = 0$.

(2) We sketch the conversion of an $\mathcal{A}$ core trace $T_P = (\kappa_P^1, \ldots, \kappa_P^n)$ on F_P into a core trace $T = (\kappa^1, \ldots, \kappa^n)$ on F, the other direction is similar. For $\mathcal{A} = \text{HS}$, every κ_P^i is a core of F_P. The corresponding core trace of F is obtained by exchanging each $\kappa_P^i = \{(\neg l_{C_i}) \mid i = 1, \ldots, n\}$ with $\kappa^i = \{C_i \mid i = 1, \ldots, n\}$. Now κ_P^i is a core of F_P iff κ^i is a core of F. To see this, note that if κ^i is not a core of F, then it can be satisfied by some assignment τ. The same τ extended by setting all l_{C_i} variables to 0 to satisfies both κ_P^i and the hard clauses $\{C_1 \vee l_{C_1}, \ldots, C_n \vee l_{C_n}\}$. Hence κ_P^i is not a core of F_P either. A similar argument shows the other direction. Finally the termination of HS after n iterations follows by a similar argument showing that $F \setminus hs$ is satisfiable for some $hs = \{C_1, \ldots, C_i\}$ iff $F^P \setminus hs^P$ is satisfiable for $hs^P = \{(\neg l_{C_1}), \ldots, (\neg l_{C_i})\}$. Hence the trace $T = (\kappa^1, \ldots, \kappa^n)$ is a HS trace on F of the same length as T_P.

For $\mathcal{A} = \text{CG}$ the argument is similar but inductive. To form a CG trace T on F, every occurrence of a $(\neg l_{C_i})$ in a clause $C^i \in \kappa_P^i$ is replaced by C_i to form a core κ^i of F^i. For $i > 0$, each such C^i may have been augmented with blocking variables, i.e., $C^i = (\neg l_{C_i} \vee \bigvee b)$ for some set of blocking variables. However, the substitution $(\neg l_{C_i} \vee \bigvee b) \to C_i \vee \bigvee b$ is still valid as, by induction, if CG adds $\bigvee b$ to $(\neg l_{C_i})$ on the execution corresponding to T_P, then it also adds $\bigvee b$ to C_i on the execution corresponding to T. $\square$

References

1. Park, J.D.: Using weighted MAX-SAT engines to solve MPE. In: Proceedings of the AAAI, pp. 682–687. AAAI Press/The MIT Press (2002)
2. Chen, Y., Safarpour, S., Veneris, A.G., Marques-Silva, J.P.: Spatial and temporal design debug using partial MaxSAT. In: Proceedings of the 19th ACM Great Lakes Symposium on VLSI, pp. 345–350. ACM (2009)
3. Chen, Y., Safarpour, S., Marques-Silva, J., Veneris, A.G.: Automated design debugging with maximum satisfiability. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **29**(11), 1804–1817 (2010)
4. Argelich, J., Berre, D.L., Lynce, I., Marques-Silva, J.P., Rapicault, P.: Solving linux upgradeability problems using boolean optimization. In: Proceedings of the LoCoCo, EPTCS, vol. 29, pp. 11–22 (2010)
5. Lynce, I., Marques-Silva, J.: Restoring CSP satisfiability with MaxSAT. Fundam. Inform. **107**(2–3), 249–266 (2011)
6. Zhu, C., Weissenbacher, G., Malik, S.: Post-silicon fault localisation using maximum satisfiability and backbones. In: Proceedings of the FMCAD, pp. 63–66. FMCAD Inc. (2011)
7. Jose, M., Majumdar, R.: Cause clue clauses: error localization using maximum satisfiability. In: Proceedings of the PLDI, pp. 437–446. ACM (2011)
8. Morgado, A., Liffiton, M., Marques-Silva, J.: MaxSAT-based MCS enumeration. In: Biere, A., Nahir, A., Vos, T. (eds.) HVC. LNCS, vol. 7857, pp. 86–101. Springer, Heidelberg (2013)
9. Guerra, J., Lynce, I.: Reasoning over biological networks using maximum satisfiability. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 941–956. Springer, Heidelberg (2012)
10. Zhang, L., Bacchus, F.: MAXSAT heuristics for cost optimal planning. In: Proceedings of the AAAI. AAAI Press (2012)

11. Ansótegui, C., Izquierdo, I., Manyà, F., Torres-Jiménez, J.: A Max-SAT-based approach to constructing optimal covering arrays. In: Proceedings of the CCIA, Frontiers in Artificial Intelligence and Applications, vol. 256, pp. 51–59. IOS Press (2013)
12. Ignatiev, A., Janota, M., Marques-Silva, J.: Towards efficient optimization in package management systems. In: Proceedings of the ICSE, pp. 745–755. ACM (2014)
13. Berg, J., Järvisalo, M., Malone, B.: Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In: Proceedings of the AISTATS, JMLR Workshop and Conference Proceedings, vol. 33, pp. 86–95 (2014). www.JMLR.org
14. Fang, Z., Li, C., Qiao, K., Feng, X., Xu, K.: Solving maximum weight clique using maximum satisfiability reasoning. In: Proceedings of the ECAI, Frontiers in Artificial Intelligence and Applications, vol. 263, pp. 303–308. IOS Press (2014)
15. Berg, J., Järvisalo, M.: SAT-based approaches to treewidth computation: an evaluation. In: Proceedings of the ICTAI, pp. 328–335. IEEE Computer Society (2014)
16. Marques-Silva, J., Janota, M., Ignatiev, A., Morgado, A.: Efficient model based diagnosis with maximum satisfiability. In: Proceedings of the IJCAI, pp. 1966–1972. AAAI Press (2015)
17. Berg, J., Järvisalo, M.: Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. Artificial Intelligence (2015, in press)
18. Wallner, J.P., Niskanen, A., Järvisalo, M.: Complexity results and algorithms for extension enforcement in abstract argumentation. In: Proceedings of the AAAI. AAAI Press (2016)
19. Li, C., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, pp. 613–631. IOS Press (2009)
20. Ansótegui, C., Bonet, M., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196**, 77–105 (2013)
21. Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: a survey and assessment. Constraints **18**(4), 478–534 (2013)
22. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the STOC, pp. 151–158. ACM (1971)
23. 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)
24. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: a partial Max-SAT solver. J. Satisfiability Boolean Model. Comput. **8**(1/2), 95–100 (2012)
25. Davies, J., Bacchus, F.: Exploiting the power of MIP solvers in MAXSAT. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 166–181. Springer, Heidelberg (2013)
26. Belov, A., Morgado, A., Marques-Silva, J.: SAT-based preprocessing for maxsat. In: Middeldorp, A., Voronkov, A., McMillan, K. (eds.) LPAR-19 2013. LNCS, vol. 8312, pp. 96–111. Springer, Heidelberg (2013)
27. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: a modular maxsat solver. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 438–445. Springer, Heidelberg (2014)
28. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Proceedings of the AAAI, pp. 2717–2723. AAAI Press (2014)
29. Bjørner, N., Narodytska, N.: Maximum satisfiability using cores and correction sets. In: Proceedings of the IJCAI, pp. 246–252. AAAI Press (2015)

30. Berg, J., Saikko, P., Järvisalo, M.: Improving the effectiveness of SAT-based preprocessing for MaxSAT. In: Proceedings of the IJCAI, pp. 239–245. AAAI Press (2015)
31. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided MaxSAT with soft cardinality constraints. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 564–573. Springer, Heidelberg (2014)
32. Ansótegui, C., Gabàs, J.: Solving (weighted) partial MaxSAT with ILP. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 403–409. Springer, Heidelberg (2013)
33. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
34. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Miller, D., Sattler, U., Gramlich, B. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
35. Lagniez, J.M., Marquis, P.: Preprocessing for propositional model counting. In: Proceedings of the AAAI, pp. 2688–2694. AAAI Press (2014)
36. 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)
37. Argelich, J., Li, C.-M., Manyà, F.: A preprocessor for Max-SAT solvers. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 15–20. Springer, Heidelberg (2008)
38. Bonet, M.L., Levy, J., Manyà, F.: Resolution for Max-SAT. Artif. Intell. **171**(8–9), 606–618 (2007)
39. Heras, F., Marques-Silva, J.: Read-once resolution for unsatisfiability-based Max-SAT algorithms. In: Proceedings of the IJCAI, pp. 572–577. AAAI Press (2011)
40. Berg, J., Saikko, P., Järvisalo, M.: Re-using auxiliary variables for maxsat preprocessing. In: Proceedings of the ICTAI, pp. 813–820. IEEE (2015)
41. Krentel, M.W.: The complexity of optimization problems. J. Comput. Syst. Sci. **36**(3), 490–509 (1988)
42. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proceedings of the AAAI. AAAI Press (2011)
43. Ignatiev, A., Morgado, A., Manquinho, V.M., Lynce, I., Marques-Silva, J.: Progression in maximum satisfiability. In: ECAI 2014, pp. 453–458. IOS Press (2014)
44. Kullmann, O., Marques-Silva, J.: Computing maximal autarkies with few and simple oracle queries (2015). CoRR abs/1505.02371
45. Janota, M., Marques-Silva, J.: On the query complexity of selecting minimal sets for monotone predicates. Artif. Intell. **233**, 73–83 (2016)
46. Ansótegui, C., Gabàs, J., Levy, J.: Exploiting subproblem optimization in SAT-based MaxSAT algorithms. J. Heuristics **22**(1), 1–53 (2016)
47. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
48. Belov, A., Järvisalo, M., Marques-Silva, J.: Formula preprocessing in MUS extraction. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 108–123. Springer, Heidelberg (2013)
49. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006)

50. 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)
51. 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)
52. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 531–548. Springer, Heidelberg (2014)
53. Bacchus, F., Narodytska, N.: Cores in core based MaxSat algorithms: an analysis. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 7–15. Springer, Heidelberg (2014)
54. Davies, J., Bacchus, F.: Postponing optimization to speed up MAXSAT solving. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 247–262. Springer, Heidelberg (2013)
55. Saikko, P., Berg, J., Järvisalo, M.: LMHS: a SAT-IP hybrid MaxSAT solver. In: Creignou, N., Le Berre, D., Le Berre, D., Le Berre, D., Le Berre, D., Le Berre, D. (eds.) SAT 2016. LNCS, vol. 9710, pp. 539–546. Springer, Heidelberg (2016). doi:10.1007/978-3-319-40970-2_34

Paper IV

Jeremias Berg, Paul Saikko, and Matti Järvisalo

Subsumed Label Elimination for Maximum Satisfiability

ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-630

Subsumed Label Elimination for Maximum Satisfiability

Jeremias Berg and **Paul Saikko** and **Matti Järvisalo**[1]

Abstract. We propose *subsumed label elimination* (SLE), a so-called *label-based* preprocessing technique for the Boolean optimization paradigm of maximum satisfiability (MaxSAT). We formally show that SLE is orthogonal to previously proposed SAT-based preprocessing techniques for MaxSAT in that it can simplify the underlying minimal unsatisfiable core structure of MaxSAT instances. We also formally show that SLE can considerably reduce the number of internal SAT solver calls within modern core-guided MaxSAT solvers. Empirically, we show that combining SLE with SAT-based preprocessing improves the performance of various state-of-the-art MaxSAT solvers on standard industrial weighted partial MaxSAT benchmarks.

1 INTRODUCTION

Maximum satisfiability (MaxSAT), the optimization counterpart of Boolean satisfiability (SAT), is becoming a competitive approach to solving hard optimization problems due to recent advances in MaxSAT solving [2, 38]. As MaxSAT is finding an increasing number of applications in solving real-world optimization problems—ranging from, e.g., inconsistency analysis, diagnosis, design debugging, and fault localization [15, 14, 4, 32, 44, 30, 39, 27, 35] to further applications in AI, combinatorics, data analysis, and bioinformatics [41, 23, 43, 3, 10, 21, 8, 9, 42]—there is a high demand for new techniques for speeding up MaxSAT solving further.

This paper focuses on improving the efficiency of solving real-world MaxSAT instances via preprocessing the instances before calling a state-of-the-art MaxSAT solver. In particular, effective preprocessing techniques for MaxSAT have the promise of providing *solver-independent* speeds-up to overall solving times, similarly to SAT where preprocessing is today an integral part of the solving process [20, 29]. This motivates work on MaxSAT-level preprocessing, in hope of bridging the gap between highly successful SAT preprocessing and the currently less studied and understood role of preprocessing for MaxSAT [7, 11, 31, 5, 13].

One approach to MaxSAT preprocessing is to lift commonly applied SAT preprocessing techniques, such as bounded variable elimination [20], self-subsuming resolution, and forms of clause elimination [26], to MaxSAT. Direct applications of such SAT preprocessing techniques are not correct w.r.t. preserving the optimal solutions of MaxSAT instances [7]. However, correct liftings to MaxSAT are enabled by the so-called *labelled conjunctive normal form* (LCNF) representation [7, 6].

A natural next goal for MaxSAT preprocessing is to go beyond lifting well-known SAT preprocessing techniques, by developing novel MaxSAT-specific LCNF-level preprocessing techniques that can be applied in conjunction with SAT-based preprocessing techniques, ideally with orthogonal simplification properties. In this paper, we address this challenge by proposing *label-based preprocessing* as a form of native LCNF-level MaxSAT preprocessing. In particular, we propose the preprocessing technique of *subsumed label elimination* (SLE). The main aim of SLE is, working in conjunction with SAT-based preprocessing on labelled MaxSAT instances, to detect and eliminate *redundant labels*, i.e., auxiliary variables that are first added to maintain correctness under SAT-based preprocessing, but which can be inferred to be redundant by a simple polynomial-time deduction rule that SLE implements. Arising from deduction rules proposed in the nineties for the so-called binate covering problem [17, 16], a key insight of SLE is that redundant labels can be eliminated by comparing the label-sets L of clauses C^L on the LCNF level, i.e., *regardless of the contents of* C. While SLE is based on a relatively simple observation, it significantly differs from the earlier proposed SAT-based preprocessing techniques for MaxSAT. In practice it also tends to provide further speed-ups to the MaxSAT solving process for several state-of-the-art MaxSAT solvers.

In more detail, we analyze how known LCNF-lifted SAT preprocessing techniques and SLE modify key properties of MaxSAT instances: the (labelled) minimal unsatisfiable cores (LMUSes) and (labelled) minimal correction sets (LMCSes). We show that SLE is fundamentally different from LCNF-lifted SAT preprocessing. In contrast to SAT preprocessing which is unable to simplify LMUSes and LMCSes, SLE can effectively remove labels from LMUSes. Via a straightforward translation of LCNFs to standard MaxSAT, this implies that SLE can reduce the number of standard MUSes in the resulting MaxSAT instance. This can improve the performance of so-called *core-guided MaxSAT solvers*, such as [22, 25, 40, 36, 37], as well as those based on the implicit hitting set approach [18, 19, 11]. Giving a concrete witnessing family of LCNF-MaxSAT instances, we show that SLE has the potential to drastically decrease the number of iterations performed by various core-guided MaxSAT solvers. Complementing the theoretical analysis, we show empirically that by combining SLE with LCNF-lifted SAT preprocessing, noticeably more labels (i.e. redundancies) are eliminated than without SLE on weighted partial MaxSAT instances of the industrial track of MaxSAT Evaluation 2015. Further, we show that the additional simplifications translate into runtime improvements for various state-of-the-art MaxSAT solvers on industrial weighted partial instances.

This paper is organized as follows. After preliminaries on labelled CNFs and SAT-based preprocessing for MaxSAT (Section 2), we detail subsumed label elimination (Section 3), and provide a theoretical analysis of SLE both in terms of its effects on the core structure of MaxSAT instances (Section 4) and its potential to speed-up MaxSAT solving (Section 5). Empirical results on simplifications provided by SLE and the impact of SLE on the performance of MaxSAT solvers are provided in Section 6.

[1] Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Finland

2 PRELIMINARIES

Throughout this paper, we work with *labelled CNFs* (LCNFs) [7, 6] which allow for generalizing MaxSAT and provide a convenient formalism for describing correct liftings of SAT preprocessing techniques to MaxSAT. For an intuitive reading, in LCNF a set of *labels* is associated with each clause. An empty label-set denotes that the corresponding clause is hard, while a non-empty label-set implies that the corresponding clause is soft. Furthermore, key concepts such as maximum satisfiability, minimal unsatisfiable subsets and minimal correction sets, are defined over the *label-sets* L of LCNF clauses C^L instead of the clauses C.

Before the formal definitions, consider the MaxSAT instance with three unweighted soft clauses shown in Figure 1 (1). As argued in [7], in order to apply e.g. bounded variable elimination (VE) [20] and still maintain the set of optimal solutions, each soft clause C_i needs to be attached an auxiliary fresh variable $\mathbf{l_i}$, resulting in the instance (Figure 1, 2a). On the level of LCNFs [6], the resulting instance is shown in Figure 1 (2b). Restricting VE from eliminating any of the added variables allows for sound application of most SAT preprocessing techniques in terms of MaxSAT. As an example, first eliminating the variable x and then y gives (possibly among others; here $\bowtie_x$ denotes resolving on x) the clause shown in Figure 1 (3a). Notice how the original one-to-one mapping between the clauses and labels vanishes, as after VE a clause may contain multiple labels. To solve the MaxSAT instance after preprocessing, the clauses obtained by preprocessing are then considered hard, and for each $\mathbf{l_i}$ the unit soft clause $(\neg\mathbf{l_i})$ with weight inherited from C_i is added into the instance. On the LCNF level, *labelled* VE [7] results equivalently in the LCNF instance (3b), explicitly separating original variables and the labels in each of the clauses.

(1) MaxSAT instance:
$C_1 = (x \vee y \vee z)$, $C_2 = (\neg x \vee \neg a \vee y)$, $C_3 = (\neg y \vee \neg a \vee \neg b)$

(2a) After adding labels: $C_1 = (x \vee y \vee z \vee \mathbf{l_1})$ $C_2 = (\neg x \vee \neg a \vee y \vee \mathbf{l_2})$ $C_3 = (\neg y \vee \neg a \vee \neg b \vee \mathbf{l_3})$ $\dots$	(3a) After variable eliminating x and y: $((C_1 \bowtie_x C_2) \bowtie_y C_3)$ $= (\neg a \vee \neg b \vee z \vee \mathbf{l_1} \vee \mathbf{l_2} \vee \mathbf{l_3})$ $\dots$
(2b) LCNF representation: $C_1^{\{\mathbf{l_1}\}}$ $C_2^{\{\mathbf{l_2}\}}$ $C_3^{\{\mathbf{l_3}\}}$ $\dots$	(3b) After labelled variable elimination on x and y: $((C_1^{\{\mathbf{l_1}\}} \bowtie_x C_2^{\{\mathbf{l_2}\}}) \bowtie_y C_3^{\{\mathbf{l_3}\}})$ $= (\neg a \vee \neg b \vee z)^{\{\mathbf{l_1},\mathbf{l_2},\mathbf{l_3}\}}$ $\dots$

Figure 1: Example of SAT-based preprocessing on the CNF and LCNF level.

2.1 Labelled CNFs and MaxSAT

Assume a countable set *Lbl* of labels. A labelled clause C^L consists of a clause C and a (possibly empty) set $L \subseteq \mathit{Lbl}$ of labels. A LCNF formula Φ is a set of labelled clauses. $Cl(\Phi)$ and $\mathit{Lbls}(\Phi)$ denote the set of clauses and labels of Φ, respectively, and $LCl(\Phi, l) = \{C^L \mid C^L \in \Phi, l \in L\}$ the set of labelled clauses in Φ that have l in their label-set. A LCNF formula is satisfiable iff $Cl(\Phi)$ (a CNF formula) is satisfiable.

Given a LCNF formula Φ and a subset $M \subseteq \mathit{Lbls}(\Phi)$ of its labels, the subformula $\Phi|_M$ of Φ induced by M is $\{C^L \in \Phi \mid L \subseteq M\}$, i.e., the LCNF formula obtained by removing from Φ all labelled clauses with at least one label not in M; notice that $\Phi|_{\mathit{Lbls}(\Phi)\setminus M} = \{C^L \in \Phi \mid L \cap M = \emptyset\}$. The *removal* REMOVE$(\Phi, K)$ of the label-set $K \subseteq \mathit{Lbls}(\Phi)$ from Φ gives $\{C^{L\setminus K} \mid C^L \in \Phi\}$, i.e, the LCNF formula obtained by removing all labels from Φ that are in K (note that removal does not remove clauses).

A (labelled) *unsatisfiable core* of an unsatisfiable LCNF formula Φ is a label-set $L \subseteq \mathit{Lbls}(\Phi)$ such that $\Phi|_L$ is unsatisfiable. An unsatisfiable core L is minimal (a LMUS) iff $\Phi|_{L'}$ is satisfiable for all $L' \subset L$. We denote the set of minimal unsatisfiable cores of Φ by LMUS(Φ). A (labelled) *minimal correction subset* (LMCS) of Φ is a label-set $R \subseteq \mathit{Lbls}(\Phi)$ such that (i) $\Phi|_{\mathit{Lbls}(\Phi)\setminus R}$ is satisfiable, and (ii) $\Phi|_{\mathit{Lbls}(\Phi)\setminus R'}$ is unsatisfiable for all $R' \subset R$. We denote the set of LMCSes of Φ by LMCS(Φ). Hitting set duality, formalizing a connection between LMUSes and LMCSes, is useful in this work.

Theorem 1 (Hitting set duality [6]) *A label-set $R \subseteq \mathit{Lbls}(\Phi)$ of a LCNF formula Φ is a LMCS of Φ iff R is an irreducible hitting set over* LMUS(Φ)*, i.e., iff R is a hitting set over* LMUS(Φ) *and no $R' \subset R$ is a hitting set of* LMUS(Φ).

A LCNF-MaxSAT instance consists of a LCNF formula Φ, and a weight function $w\colon \mathit{Lbls}(\Phi) \to \mathbb{N}$ assigning a positive weight $w(l)$ to each label $l \in \mathit{Lbls}(\Phi)$. The cost of a label-set $L \subseteq \mathit{Lbls}(\Phi)$ is the sum of the weights of the labels in L. Given a LCNF-MaxSAT instance Φ such that $\Phi|_\emptyset$ is satisfiable, any assignment τ that satisfies $\Phi|_\emptyset$ is a solution to the LCNF-MaxSAT instance. A solution τ is optimal if it satisfies $\Phi|_{\mathit{Lbls}(\Phi)\setminus R}$ for some minimum-cost LMCS R of Φ. The cost of τ is the cost of R. We treat the MaxSAT problem for LCNFs as the problem of computing R. In the rest of the text we will always assume that solutions to (Φ, w) exist, i.e., that $\Phi|_\emptyset$ is satisfiable.

A (standard/non-labelled) MaxSAT instance $F = (F_h, F_s, w)$ consists of a set F_h of hard and a set F_s of soft clauses, together with a function $w\colon F_s \to \mathbb{N}$ assigning a positive weight $w(C)$ to each soft clause $C \in F_s$. A (standard) minimal correction set (MCS) of F is a subset-minimal subset of F_s whose removal from F_s makes the instance satisfiable. Similarly, a (standard) minimal unsatisfiable core (MUS) of F is a subset-minimal subset F_s' for which $F_h \cup F_s'$ is an unsatisfiable set of clauses. Given a non-labelled MaxSAT instance F, any truth assignment τ satisfying all hard clauses is a solution to the instance. A solution τ is optimal if the sum of the weights of the soft clauses τ satisfies is the maximum over all solutions. Notice that the soft clauses falsified by an optimal solution form a minimum-cost MCS of F.

A MaxSAT instance $F = (F_h, F_s, w)$ can be viewed as a LCNF-MaxSAT instance (Φ_F, w) by introducing (i) for each hard clause $C \in F_h$ the labelled clause $C^\emptyset$, and (ii) for each soft clause $C \in F_s$ the labelled clause $C^{\{l_C\}}$, where l_C is a distinct label for C with weight $w(l_C) = w(C)$. It is easy to see that any optimal solution to Φ_F is an optimal solution to F, and vice versa. An essential intuition is that LMCSes of (Φ_F, w) correspond exactly to the MCSes of (F_h, F_s, w) in that for any MCS $\{C_1, \dots, C_k\}$ there is a corresponding LMCS $\{l_{C_1}, \dots, l_{C_k}\}$ (and vice versa). Similarly, LMUSes of (Φ_F, w) correspond to MUSes of (F_h, F_s, w).

To the other direction, a LCNF-MaxSAT instance (Φ, w) can be viewed as a MaxSAT instance F_Φ by associating with each label $l_i \in \mathit{Lbls}(\Phi)$ a distinct variable a_i, and introducing (i) for each labelled clause $C^L \in \Phi$ a hard clause $C \vee \bigvee_{l_i \in L} a_i$, and (ii) for each $l_i \in \mathit{Lbls}(\Phi)$, a soft clause $(\neg a_i)$ with weight $w((\neg a_i)) = w(l_i)$, where $w(l_i)$ is the weight of the label l_i. Again, using this reduction, LMUSes and LMCSes of (Φ, w) correspond exactly to the MUSes and MCSes of F_Φ. Importantly for this work, especially the discussion in Section 5, this reduction allows one to treat any standard MaxSAT solver as a LCNF-MaxSAT solver.

Example 2 *Consider the MaxSAT instance $F_{\text{ex}} = (F_h, F_s, w)$ with $w(C) = 1$ for all $C \in F_s$, $F_h = \{(x \vee y), (\neg t \vee \neg z), (\neg z \vee y), (\neg y \vee z), (z \vee t)\}$, and $F_s = \{(\neg x), (x), (y \vee t), (z \vee t \vee x)\}$. The assignment τ for which $\tau(t) = \tau(x) = 0$ and $\tau(y) = \tau(z) = 1$ is an optimal solution to F_{ex} with cost 1. The LCNF-MaxSAT instance $\Phi_{F_{\text{ex}}}$ corresponding to F_{ex} is*

$$\Phi_{F_{\text{ex}}} = \{(x \vee y)^\emptyset, (\neg t \vee \neg z)^\emptyset, (\neg z \vee y)^\emptyset, (\neg y \vee z)^\emptyset, (z \vee t)^\emptyset,$$
$$(\neg x)^{\{l_1\}}, (x)^{\{l_2\}}, (y \vee t)^{\{l_3\}}, (z \vee t \vee x)^{\{l_4\}}\}$$

with $w(l_i) = 1$ for $i = 1..4$. Now $Cl(\Phi_{F_{\text{ex}}}) = F_h \cup F_s$ and $Lbls(\Phi_{F_{\text{ex}}}) = \{l_1, l_2, l_3, l_4\}$. The label-set $L = \{l_1, l_2\}$ is an LMUS of $\Phi_{F_{\text{ex}}}$ as

$$\Phi_{F_{\text{ex}}}|_L = \{(x \vee y)^\emptyset, (\neg t \vee \neg z)^\emptyset, (\neg z \vee y)^\emptyset, (\neg y \vee z)^\emptyset,$$
$$(z \vee t)^\emptyset, (\neg x)^{\{l_1\}}, (x)^{\{l_2\}}\}$$

is unsatisfiable. The sets $R_1 = \{l_1\}$ and $R_2 = \{l_2\}$ are examples of (minimum-cost) LMCSes of $\Phi_{F_{\text{ex}}}$. The fact that τ is an optimal solution to the LCNF-MaxSAT instance $\Phi_{F_{\text{ex}}}$ can be verified by checking that τ satisfies $\Phi_{F_{\text{ex}}}|_{Lbls(\Phi_{F_{\text{ex}}}) \setminus R_2}$. Converting $\Phi_{F_{\text{ex}}}$ back to MaxSAT results in the instance $F' = (F_h', F_s', w)$ with

$$F_h' = \{(x \vee y), (\neg t \vee \neg z), (\neg z \vee y), (\neg y \vee z), (z \vee t),$$
$$(\neg x \vee a_1), (x \vee a_2), (y \vee t \vee a_3), (z \vee t \vee x \vee a_4)\}$$
and $$F_s' = \{(\neg a_1), (\neg a_2), (\neg a_3), (\neg a_4)\}.$$

2.2 SAT-based Preprocessing for LCNFs

A motivation for viewing MaxSAT instances as LCNF in [7] was to develop sound applications of SAT preprocessing techniques for MaxSAT. Many important SAT preprocessing techniques, including bounded variable elimination (VE) [20], self-subsuming resolution (SSR), and subsumption elimination (SE), cannot be used directly on MaxSAT instances [7]. However, the techniques can be applied on LCNFs by taking into account the natural restrictions implied by the SAT-level techniques on the label-sets of labelled clauses. With this intuition, the following LCNF-liftings of VE, SSR, and SE were proposed [7].

- *LCNF-lifting of the resolution rule:* The resolvent of two labelled clauses $(x \vee A)^{L_1}$ and $(\neg x \vee B)^{L_2}$ w.r.t. x is $(x \vee A)^{L_1} \bowtie_x (\neg x \vee B)^{L_2} = (A \vee B)^{L_1 \cup L_2}$.
- *LCNF-lifting of VE (LVE):* Let Φ_x and $\Phi_{\neg x}$, resp., denote the sets of labelled clauses that contain the literal x and the literal $\neg x$, resp. LVE allows for replacing $\Phi_x \cup \Phi_{\neg x}$ with $\Phi_x \bowtie_x \Phi_{\neg x} = \{A^{L_1} \bowtie_x B^{L_2} \mid A^{L_1} \in \Phi_x, B^{L_2} \in \Phi_{\neg x}, A \vee B$ non-tautological$\}$ given that $|\Phi_x \bowtie_x \Phi_{\neg x}| \leq |\Phi_x \cup \Phi_{\neg x}|$.
- *LCNF-lifting of SE (LSE):* A labelled clause A^{L_1} subsumes B^{L_2} if $A \subseteq B$ *and* $L_1 \subseteq L_2$. LSE allows for removing subsumed clauses.
- *LCNF-lifting of SSR (LSSR):* Given labelled clauses $(l \vee A)^{L_1}$ and $(\neg l \vee B)^{L_2}$, if A^{L_1} subsumes B^{L_2}, LSSR allows for replacing $(\neg l \vee B)^{L_2}$ with B^{L_2}.

Blocked clause elimination (BCE) [28] is sound for MaxSAT [7], and could as such be directly applied on MaxSAT instances. However, for a uniform presentation, it makes sense to consider a straightforward lifting of BCE.

- *LCNF-lifting of BCE (LBCE):* A labelled clause C^L is *blocked* in Φ if C is blocked in $Cl(\Phi)$. LBCE allows for removing blocked clauses.

Example 3 *Consider the LCNF-MaxSAT instance $\Phi_{F_{\text{ex}}}$ from Example 2. Applying LSE to remove $(z \vee t \vee x)^{\{l_4\}}$ and LVE to eliminate x and t results in the formula*

$$\{(y)^{\{l_1\}}, (\neg z \vee y)^\emptyset, (\neg y \vee z)^\emptyset, ()^{\{l_1, l_2\}}, (y \vee \neg z)^{\{l_3\}}\}.$$

Removing $(y \vee \neg z)^{\{l_3\}}$ by LSE and eliminating z by LVE results in the preprocessed formula $\Phi_{F_{\text{ex}}}^{pre} = \{(y)^{\{l_1\}}, ()^{\{l_1, l_2\}}\}$.

LVE, LSSR, LSE, and LBCE are correct due to the following.

Proposition 4 ([7]) *Let Φ be a LCNF-MaxSAT instance and Φ^{pre} the LCNF-MaxSAT instance resulting from an application of LVE, LSSR, LSE, and LBCE on Φ. Then $\text{LMUS}(\Phi) = \text{LMUS}(\Phi^{pre})$ and, by Theorem 1, $\text{LMCS}(\Phi) = \text{LMCS}(\Phi^{pre})$.*

3 SUBSUMED LABEL ELIMINATION

We propose and analyze *subsumed label elimination* (SLE), a *label-based preprocessing* technique for MaxSAT. The primary goal of SLE is to provide further simplifications when applied in conjunction with SAT-based preprocessing; SLE focuses on removing labels from *non-singleton* label-sets (produced starting from non-labelled MaxSAT instances mainly by LVE). Before a formal definition of SLE, we begin with an example to illustrate some of the shortcomings of SAT-based preprocessing for MaxSAT that SLE seeks to address.

Example 5 *Consider the MaxSAT instance $F = (F_h, F_s, w)$ with $w(C) = 1$ for all $C \in F_s$ and*

$$F_h = \{(x \vee y)\} \text{ and } F_s = \{(\neg x), (\neg y)\}.$$

Converting F to LCNF gives the instance $\Phi_F = \{(x \vee y)^\emptyset, (\neg x)^{\{l_1\}}, (\neg y)^{\{l_2\}}\}$. Applying LVE to eliminate both x and y results in the LCNF-MaxSAT instance $pre(\Phi_F) = \{()^{\{l_1, l_2\}}\}$. Finally, converting $pre(\Phi_F)$ back to MaxSAT gives the MaxSAT instance $F' = (F_h', F_s', w)$ with

$$F_h' = \{(a_1 \vee a_2)\} \text{ and } F_s' = \{(\neg a_1), (\neg a_2)\},$$

i.e., the exact same instance as F modulo variable naming. In other words, LVE (or LSSR, LSE, and LBCE) is unable to simplify F. Furthermore, notice that F contains exactly one MUS: $\{(\neg x), (\neg y)\}$. As the clauses $(\neg x)$ and $(\neg y)$ occur in exactly the same MUSes, no optimal solution to F falsifies both of them. As an alternative view, no MCS of F contains both $(\neg x)$ and $(\neg y)$, which means that either clause could be hardened, *i.e., changed to a hard clause, without removing all of the optimal solutions of the instance. As we will see, SLE captures this simplification on the LCNF-level.*

More concretely, consider a LCNF-MaxSAT instance Φ. SLE is based on the following observation. Consider two labels $l_1, l_2 \in Lbls(\Phi)$ such that $w(l_1) \leq w(l_2)$, and l_1 appears in at least the same LMUSes of Φ as l_2. Then l_2 is redundant in that l_2 can be replaced by l_1 in any LMCS R of Φ without increasing the cost of R. Hence l_2 can be removed from Φ while maintaining at least one minimum-cost LMCS. This is more formally stated as Theorem 6.

Theorem 6 *Let $l_1, l_2 \in Lbls(\Phi)$ and $\Phi^{pre} = \text{REMOVE}(\Phi, \{l_2\})$. Assume that, for all $L \in \text{LMUS}(\Phi)$, $l_2 \in L$ implies $l_1 \in L$. Then $\emptyset \neq \text{LMCS}(\Phi^{pre}) \subseteq \text{LMCS}(\Phi)$.*

Proof. $\Phi^{pre}|_{Lbls(\Phi^{pre})\setminus R} = \Phi|_{Lbls(\Phi)\setminus R}$ for any label-set $R \subseteq Lbls(\Phi^{pre})$. Hence it suffices to show that there is an $R \in \text{LMCS}(\Phi)$ s.t. $R \subseteq Lbls(\Phi^{pre})$. This can be verified by viewing R as an irreducible hitting set of LMUS(Φ). If $R \not\subseteq Lbls(\Phi^{pre})$, then $l_2 \in R$. By assumption, $R' = (R \setminus \{l_2\}) \cup \{l_1\}$, a subset of $Lbls(\Phi^{pre})$, is also an irreducible hitting set of LMUS(Φ) and hence a LMCS of Φ. □

While the assumption in Theorem 6 is likely not checkable in polynomial time, a stricter, easier-to-check version of the assumption, formalized in Proposition 7, gives the basis for SLE. In words, let L be any label-set and $C^{L'}$ any labelled clause of Φ. If L' contains labels l_1 and l_2 such that $l_2 \in L$ but $l_1 \notin L$, then $C^{L'}$ is not a member of the formula $\Phi|_L$. This is specifically true for any LMUS of Φ.

Proposition 7 *Let $l_1, l_2 \in Lbls(\Phi)$ and $LCl(\Phi, l_2) \subseteq LCl(\Phi, l_1)$. Then, for all $L \in \text{LMUS}(\Phi)$, $l_2 \in L$ implies $l_1 \in L$.*

Proof. Let L be a label-set such that $l_2 \in L$ and $l_1 \notin L$. We show that L is not a LMUS of Φ. From the assumption $LCl(\Phi, l_2) \subseteq LCl(\Phi, l_1)$ it follows that, if $C^{L'}$ is a labelled clause for which $l_2 \in L'$, then $l_1 \in L'$. Thus $C^{L'} \notin \Phi|_L$, and hence $\Phi|_L = \Phi|_{L\setminus\{l_2\}}$. As such $L \notin \text{LMUS}(\Phi)$ as either $\Phi|_L$ is satisfiable or $\Phi|_{L_1}$ is unsatisfiable for $L_1 = L \setminus \{l_2\} \subset L$. □

The final part in the formalization of SLE ensures that the removal of l_2 preserves at least one *minimum-cost* LMCS of the instance. This follows by adding an assumption on the weights of l_1 and l_2.

Proposition 8 *Let $l_1, l_2 \in Lbls(\Phi)$ and $\Phi^{pre} = \text{Remove}(\Phi, \{l_2\})$. Assume that, for all $L \in \text{LMUS}(\Phi)$, $l_2 \in L$ implies $l_1 \in L$, and $w(l_1) \leq w(l_2)$. Then all minimum-cost LMCSes of Φ^{pre} are also minimum-cost LMCSes of Φ.*

Proof. Following the proof of Theorem 6 let $R' = (R\setminus\{l_2\}) \cup \{l_1\}$ be the LMCS of Φ constructed in order to replace the LMCS $R \not\subseteq Lbls(\Phi^{pre})$. The extra assumption on the weights guarantees that the cost of R' is not higher than the cost of R. □

Putting these results together gives SLE. Informally, SLE removes subsumed labels l_2, or, more formally, converts Φ into $\text{Remove}(\Phi, \{l_2\})$.

Definition 9 (Subsumed Label Elimination (SLE)) *Let Φ be a LCNF-MaxSAT instance and $l_1, l_2 \in Lbls(\Phi)$. We say that l_1 subsumes l_2 if (i) $LCl(\Phi, l_2) \subseteq LCl(\Phi, l_1)$, and (ii) $w(l_1) \leq w(l_2)$. SLE allows for removing subsumed labels from LCNF-MaxSAT instances.*

Example 10 *Consider the LCNF-MaxSAT instance*

$$\begin{aligned}\Phi =& \{(x_i \vee y_j)^{\emptyset} \mid i, j = 1..4\} \cup \\ & \{(\neg x_i \vee \neg x_3)^{\emptyset}, (\neg x_i \vee \neg x_4)^{\emptyset} \mid i = 1, 2\} \cup \\ & \{(\neg y_i)^{\{l, l_i\}}, (\neg y_i)^{\{l, t_i\}} \mid i = 1..4\}\end{aligned}$$

with $w(l) = 1$ and $w(l_i) = w(t_i) = 2$ for all i. First note that LVE, LSSR, LSE, and LBCE cannot simplify Φ. Specifically, as every variable appears both negatively and positively at least twice and no produced resolvents are tautologies, LVE cannot eliminate any variables. However, l subsumes all of the other labels, and hence applying SLE gives

$$\begin{aligned}& \{(x_i \vee y_j)^{\emptyset} \mid i, j = 1..4\} \cup \\ & \{(\neg x_i \vee \neg x_3)^{\emptyset}, (\neg x_i \vee \neg x_4)^{\emptyset} \mid i = 1, 2\} \cup \\ & \{(\neg y_i)^{\{l\}} \mid i = 1..4\}.\end{aligned}$$

Each y_i appears negatively only in a single clause and can hence be eliminated by LVE, resulting in

$$\{(x_i)^{\{l\}} \mid i = 1..4\} \cup \{(\neg x_i \vee \neg x_3)^{\emptyset}, (\neg x_i \vee \neg x_4)^{\emptyset} \mid i = 1, 2\}.$$

Now each x_i only appears positively in a single clause. LVE then gives $\Phi^{pre} = \{()^{\{l\}}\}$.

Remark 1 *While the main focus of this work is on understanding the effect of SLE on the core structure of MaxSAT instances and the potential of SLE to speed up state-of-the-art MaxSAT solvers, we note that SLE (for MaxSAT) can be viewed as the counterpart of the so-called* dominance rule *proposed in the early 90s in conjunction with branch-and-bound approaches for the so-called* binate covering problem *[17, 16] with applications in logic synthesis. More details on this connection are provided in Appendix A. To the best of our knowledge, however, SLE has not been previously proposed, analyzed, or empirically evaluated in the context of MaxSAT.*

4 EFFECTS OF SLE

We continue by analyzing SLE in terms of how it simplifies LCNFs. We show that SLE is orthogonal to the LCNF-lifted SAT-based preprocessing techniques in terms of the LMUSes and LMCSes—and hence MaxSAT solutions—preserved under simplification.

We start with relatively simple corollaries of the definition. First, we observe that subsumed labels remain subsumed after applications of SAT-based preprocessing.

Proposition 11 *Let $l \in Lbls(\Phi)$ and assume that SLE can eliminate l from Φ. Let Φ^{pre} be Φ after applying LVE, LSSR, LSE, or LBCE. Then SLE can eliminate l from Φ^{pre}.*

Proof. Let l_1 be a label that subsumes l in Φ. It suffices to show that the preconditions of SLE are satisfied in Φ^{pre}. First, the precondition $w(l_1) \leq w(l)$ is trivially satisfied as none of the techniques alter the weights of labels. For the second precondition, $LCl(\Phi^{pre}, l) \subseteq LCl(\Phi^{pre}, l_1)$, the non-trivial case is $LCl(\Phi^{pre}, l) \neq \emptyset$. As $LCl(\Phi, l) \subseteq LCl(\Phi, l_1)$, it is enough to verify that none of the SAT-based preprocessing techniques introduce a labelled clause $C^L \in \Phi^{pre}$ with $l \in L$ and $l_1 \notin L$. This is trivially true for LSE and LBCE as they only remove clauses. This is also true for LSSR as it only removes literals, not labels. Finally, LVE cannot produce resolvents which contain l but not l_1, since there are no labelled clauses $C^{L'}$ in Φ with $l \in L'$ and $l_1 \notin L'$. Thus the label-set of any resolvent produced by LVE, which is a union of label-sets in Φ, contains either both or neither of l_1 and l. □

Thus it makes sense to incorporate SLE into the preprocessing loop together with LVE, LSSR, LSE, and LBCE.

In analogy with Proposition 11, subsumed labels remain subsumed also under SLE steps quite generally. An exception comes from cases in which two labels l_1 and l_2 subsume each other, i.e., when l_1 and l_2 occur in exactly the same label-sets *and* $w(l_2) = w(l_1)$. Note also that, generally, if l_1 subsumes l_2, and l_2 subsumes l_3, then l_1 subsumes l_3.

Turning to comparing SLE and SAT-based preprocessing, Propositions 4 and 12 together illustrate fundamental differences between SLE and LVE, LSSR, LSE, and LBCE. By Proposition 4, LVE, LSSR, LSE, and LBCE preserve the LMUSes of LCNF-MaxSAT instances. This is not true for SLE. Instead, SLE guarantees (only) that at least one minimum-cost (optimal) LMCS and, as such, that at least one optimal solution of the instance is preserved.

Proposition 12 *SLE does not in general preserve LMUSes (or LMCSes) of LCNF-MaxSAT instances.*

Proof. Consider the instances Φ and Φ^{pre} from Example 10. The sets $\{l, l_i\}$ and $\{l, t_i\}$ are LMUSes of Φ for all i but not of Φ^{pre}. $\square$

An alternative way of stating Proposition 12 is that applying SLE does not in general preserve all optimal solutions to LCNF-MaxSAT instances. For a simple example, consider the LCNF-MaxSAT instance $\Phi = \{(x)^{\{l_1\}}, (\neg x)^{\{l_2\}}\}$ with unit-weighted labels. There are two optimal solutions to Φ: $\tau_1(x) = 1$ satisfying $\Phi|_{Lbls(\Phi)\setminus\{l_2\}}$, and $\tau_2(x) = 0$ satisfying $\Phi|_{Lbls(\Phi)\setminus\{l_1\}}$. However, by LVE we can simplify Φ to $\{()^{\{l_1,l_2\}}\}$ and by SLE further to $\{()^{\{l_1\}}\}$. The only LMCS of the simplified instance is $\{l_1\}$, corresponding to the solution τ_2.

Instead of preserving LMUSes, SLE could be seen as a form of LMUS minimization in the sense that all LMUSes remaining after SLE are projections of LMUSes of the original LCNF onto the remaining set of labels.

Theorem 13 *Let Φ be a LCNF-MaxSAT instance and $l \in Lbls(\Phi)$ a subsumed label. Let $\Phi^{pre} =$ REMOVE$(\Phi, \{l\})$, i.e., the formula after eliminating l by SLE from Φ. Then all LMUSes L^p of Φ^{pre} are of the form $L^p = L \cap Lbls(\Phi^{pre})$ for some LMUS L of Φ.*

Proof. First notice that $\Phi|_{L^p} \subseteq \Phi^{\text{pre}}|_{L^p}$ as the restriction operator only removes labels from label-sets, not clauses. If $\Phi|_{L^p} = \Phi^{\text{pre}}|_{L^p}$, then the same will be true for any $L^p_s \subseteq L^p$, so L^p itself is an LMUS of Φ. Otherwise, the reason for a labelled clause C^L to be in $\Phi^{\text{pre}}|_{L^p}$ but not in $\Phi|_{L^p}$ is that the eliminated label l was in L, i.e., $C^L \notin \Phi$ but $C^{L\cup\{l\}} \in \Phi$. Hence $\Phi|_{L^p\cup\{l\}} = \Phi^{\text{pre}}|_{L^p}$, and $L^p \cup \{l\}$ is a LMUS of Φ. $\square$

For further differences between SLE and LVE, LSSR, LSE, and LBCE, consider a MaxSAT instance F and a soft clause $C \in F_s$. Let Φ_F be the LCNF-MaxSAT instance corresponding to F and l_C the label for which $C^{\{l_C\}} \in \Phi_F$. A simple application of Theorem 4 gives that if l_C is removed from Φ_F by LVE, LSSR, LSE, or LBCE, then *any* optimal solution to Φ_F, which is also an optimal solution to F, will satisfy C.

Proposition 14 *Let Φ^{pre}_F be the instance resulting after an application of LVE, LSSR, LSE, or LBCE on Φ_F. If $l_C \notin Lbls(\Phi^{pre}_F)$, then any optimal solution τ to Φ_F, which is also an optimal solution to F, will satisfy C.*

Proof. Since τ is optimal, it satisfies $\Phi_F|_{Lbls(\Phi_F)\setminus R}$ for some minimum-cost LMCS R of Φ_F. By Theorem 4, $l_C \notin R$, and thus $C \in Cl(\Phi_F|_{Lbls(\Phi_F)\setminus R})$. $\square$

Informally, it could be said that SAT-based preprocessing can only remove labels that are "uninteresting" in terms of LMCS computation. In contrast, elimination of l_C by SLE means that *some* (but not necessarily all) optimal solutions of F satisfy C, as shown next.

Proposition 15 *Let Φ^{pre}_F be the instance resulting from an application of SLE on Φ_F. If $l_C \notin Lbls(\Phi^{pre})$, then there is an optimal solution τ to Φ_F and F that satisfies C. Furthermore, there may exist optimal solutions to Φ_F that do not satisfy C.*

Proof. By the assumption that l_C is subsumed, it follows from Theorem 6 and Proposition 8 that there is a minimum-cost LMCS R of Φ_F for which $l_C \notin R$. The first part of the claim follows by observing that $\Phi_F|_{Lbls(\Phi_F)\setminus R}$ is satisfiable and $C \in Cl(\Phi_F|_{Lbls(\Phi_F)\setminus R})$. For the second part of the claim, consider the discussion following Proposition 12. $\square$

5 SLE AND CORE-GUIDED SOLVERS

We now show that SLE has the potential to considerably lower the number of iterations made by so-called *core-guided MaxSAT solvers*, one of the most successful current MaxSAT solving approaches. The core-guided approach has several variants, e.g. [2, 38, 22, 25, 40, 36, 37, 18, 19]. In this work, we study the effect of SLE on two different types of core-guided solvers through generic abstractions. The first one, *CG-MaxSAT* (Algorithm 1), iteratively employs a SAT solver to extract unsatisfiable cores and rules out each of the found cores from the formula by a *clause replication and relaxation* step. Several algorithms that fit the CG-MaxSAT abstraction have been proposed [22, 25, 40, 36, 37]. The second one, MaxHS (Algorithm 2), is an abstraction of the implicit hitting set approach to MaxSAT [18, 19], iteratively using a SAT solver to extract unsatisfiable cores, and an exact minimum-cost hitting set algorithm to compute hitting sets over the found cores.

In more detail, at each iteration i, CG-MaxSAT invokes a SAT solver on the clauses of a working formula F^i_w (initialized as all clauses of the MaxSAT instance viewed as hard). If the working formula is satisfiable, CG-MaxSAT terminates and returns the satisfying assignment returned by the SAT solver. Otherwise, the SAT solver returns an unsatisfiable core κ of F^i_w. CG-MaxSAT then duplicates the clauses in κ to create two sets κ^r and $\kappa^{\bar{r}}$. Both sets contain exactly the same clauses as κ; each clause $C \in \kappa$ is duplicated into two: $C^r \in \kappa^r$ and $C^{\bar{r}} \in \kappa^{\bar{r}}$. The weight of C^r is set to w_m, the minimum weight over the clauses in the core, and the weight of $C^{\bar{r}}$ to $w(C) - w_m$. The clauses of $\kappa^{\bar{r}}$ are added to the working formula unaltered. Finally, the working formula is updated by *relaxing* the clauses in κ^r. The method of relaxation varies between core-guided solvers. For our analysis, the important consequences of relaxation are that the (possibly altered) clauses of κ^r do not appear as a core in future iterations, and that the optimal cost of F^{i+1}_w (when viewed as a MaxSAT instance) is exactly w_m lower than the optimal cost of F^i_w. Termination of CG-MaxSAT is guaranteed by the fact that $w_m > 0$ on all iterations and that a MaxSAT instance of cost 0 is satisfiable as a SAT instance. For a concrete example of a relaxation step, consider the classical Fu-Malik algorithm [22] and its extensions to the weighted case [33, 1]. These algorithms augments each $C_i \in \kappa^r$ with a fresh relaxation variable r_i, creating the clause $C_i \vee r_i$, and additionally adds a hard *exactly-one* constraint $\sum r_i = 1$ over the relaxation variables. The intuition behind this step is that assigning a relaxation variable to 1 effectively removes the corresponding clause from the formula, hence removing the core κ^r. Additionally,

```
Input: MaxSAT instance $F = (F_h, F_s, w)$.
Output: An optimal solution $\tau$ for $F$.
$F^0_w \leftarrow F_h \cup F_s$
for i=0,... do
    $(result, \kappa, \tau) \leftarrow$ SATSOLVE$(F^i_w)$
    if result="satisfiable" then
        return $\tau$                          // optimal solution
    else
        $F^i_w = (F^i_w \setminus \kappa)$          // SAT solver returned unsat core
        $w_m \leftarrow \min\{w(C) \mid C \in \kappa\}$
        $(\kappa^r, \kappa^{\bar{r}}) \leftarrow$ CLAUSEREPLICATE$(\kappa, w_m)$
        $F^i_w \leftarrow F^i_w \cup \kappa^{\bar{r}}$
        $F^{i+1}_w \leftarrow$ RELAX$(F^i_w, \kappa^r)$
    end
end
```

Algorithm 1: CG-MaxSAT

```
Input: MaxSAT instance $F = (F_h, F_s, w)$.
Output: An optimal solution $\tau$ for $F$.
$\mathcal{K} \leftarrow \emptyset$                    // set of found unsat cores of $F$
$F_w \leftarrow (F_h \cup F_s)$
while true do
    $H \leftarrow$ MinCostHittingSet($\mathcal{K}$)
    $F_w \leftarrow F_h \cup (F_s \setminus H)$
    (result, $\kappa, \tau$) $\leftarrow$ SATSolve($F_w$)
    if result="satisfiable" then
        return $\tau$                                  // optimal solution
    else
        $\mathcal{K} \leftarrow \mathcal{K} \cup \{\kappa\}$          // SAT solver returned unsat core
    end
end
```

Algorithm 2: MaxHS

the exactly-one constraint ensures that the cost is lowered exactly by w_m.

MaxHS is a hybrid algorithm that uses a SAT solver for core extraction over a working formula F_w (initialized as all clauses of the input instance viewed as hard). Given a collection $\mathcal{K}$ of extracted cores, MaxHS uses an exact algorithm (integer programming solver in practice) to find a minimum-cost hitting set hs over $\mathcal{K}$. The working formula is then updated to contain all clauses of F except for the soft clauses in hs, and the SAT solver is invoked again. If the working formula is satisfiable, the satisfying assignment obtained is an optimal solution to F. Otherwise another core is obtained and the search continues again with hitting set computation.

The main result of this section is that there are families of LCNF-MaxSAT instances on which SLE can significantly decrease the number of SAT solver calls and clause replication when subsequently solving the instances with CG-MaxSAT or MaxHS.

Proposition 16 *For $\mathcal{A} \in \{$CG-MaxSAT, MaxHS$\}$, there is a family of LCNF-MaxSAT instances Φ_N, with $\Theta(N)$ different labels, on which*

(i) $\mathcal{A}$ requires $\Theta(N)$ calls to its SAT solver, and, for $\mathcal{A}$ = CG-MaxSAT, $\mathcal{A}$ requires $\Theta(N)$ clause replication steps, on $\Theta(N!)$ different executions; while
(ii) $\mathcal{A}$ is guaranteed to require only two (one unsatisfiable and one satisfiable) SAT solver calls if SLE is applied on Φ_N before $\mathcal{A}$

under the assumption that the internal SAT solver is guaranteed to return minimal unsatisfiable cores.

Proof. The family of LCNF-MaxSAT instances witnessing the claim is the same for CG-MaxSAT and MaxHS. Let N be sufficiently large and define

$$\Phi_N := \bigcup_{i=1}^{2N-2} \mathbf{P}_i \cup \bigcup_{i=1}^{N-1} \mathbf{H}_i, \text{ where}$$

$$\mathbf{P}_i = \bigcup_{j=1}^{N} \{(\neg p_i^j \vee \neg p_k^j)^{\emptyset} \mid k = (i+1)..(2N-1)\} \text{ and}$$

$$\mathbf{H}_i = \left\{ \left(\bigvee_{j=1}^{N} p_k^j \right)^{\{l, l_i\}} \;\middle|\; k = i..(N+i) \right\},$$

with $w(l) = w(l_{N-1}) = N$ and $w(l_i) = 1$ for all other labels l_i. Notice that Φ_N contains $N-1$ LMUSes of the form $\{l, l_i\}$ for all $1 \leq i \leq N-1$. Hence, the only minimum-cost LMCS of Φ_N is $\{l\}$. Furthermore, refuting any of the LMUSes requires proving the unsatisfiability of the formula $\Phi_N|_{\{l,l_i\}}$, which corresponds to an instance of the pigeonhole principle; meaning that the extraction any of the LMUSes of Φ_N requires an exponentially long SAT solver call [24]. Next we sketch the executions of both CG-MaxSAT and MaxHS that require $\Theta(N)$ SAT-solver calls when solving Φ_N.

Conversion of Φ_N to MaxSAT results in the formula $F = (F_h, F_s, w)$, where

$$F_h = \bigcup_{i=1}^{2N-2} \bigcup_{j=1}^{N} \{(\neg p_i^j \vee \neg p_k^j) \mid k = i..(2N-1)\}$$
$$\cup \bigcup_{i=1}^{N-1} \left\{ \left(a_l \vee a_i \vee \bigvee_{j=1}^{N} p_k^j \right) \;\middle|\; k = i..(N+i) \right\}$$

and $F_s = \{(\neg a_l), (\neg a_1), \ldots, (\neg a_{N-1})\}$ with $w((\neg a_l)) = w((\neg a_{N-1})) = N$ and $w(C) = 1$ for all other $C \in F_s$. The MUSes of F correspond exactly to the LMUSes of Φ_N and are of the form $\{(\neg a_l), (\neg a_i)\}$ for all $i = 1..N-1$. For an intuition on the executions requiring a linear number of SAT solver calls of both algorithms, notice that both can terminate immediately and only after encountering and processing the MUS $\{(\neg a_l), (\neg a_{N-1})\}$ corresponding to the the LMUS $\{l, l_{N-1}\}$.

For $\mathcal{A}$ = MaxHS, assume that the internal SAT solver returns the MUSes of in any order with $\{(\neg a_l), (\neg a_{N-1})\}$ last. Then the hitting set hs computed by MaxHS will not contain the clause $(\neg a_l)$ before the $(N-1)$th iteration and as such MaxHS can not terminate as $F \setminus hs$ will always contain the MUS $\{(\neg a_l), (\neg a_{N-1})\}$. There are a total of $(N-2)!$ executions in which the MUS $\{(\neg a_l), (\neg a_{N-1})\}$ is returned last.

For $\mathcal{A}$ = CG-MaxSAT, the long executions are similar. Assume that the first MUS returned by the SAT-solver in CG-MaxSAT is $\{(\neg a_l), (\neg a_1)\}$. The smallest weight w_m of the clauses in the core is 1, so CG-MaxSAT proceeds by replicating the clause $(\neg a_l)$ into two clauses $C^r = (\neg a_l)$ and $C_2 = (\neg a_l)$, setting $w(C^r) = 1$ and $w(C_2) = N-1$, adding C_2 back into the working formula, relaxing the core $\{C^r, (\neg a_1)\}$, and reiterating. Assume that CG-MaxSAT proceeds similarly by processing the cores $\{(\neg a_l)^i, (\neg a_i)\}$ for $i = 1..N-2$ during the first $N-2$ iterations where $(\neg a_l)^i$ is the copy of the clause $(\neg a_l)$ produced in the previous iteration. Finally on the $(N-1)$th iteration CG-MaxSAT encounters the core $\{(\neg a_l)^{N-2}, (\neg a_{N-1})\}$. At this point $w((\neg a_l)^{N-2}) = 2$ and $w((\neg a_{N-1})) = N$, so CG-MaxSAT replicates $(\neg a_{N-1})$ and relaxes the core before invoking its SAT solver one final time in order to find the current working formula satisfiable. In total, CG-MaxSAT performs N SAT solver calls and $N-1$ clause replications. A similar argument can be made for any ordering of the MUSes with $\{(\neg a_l), (\neg a_{N-1})\}$ last.

Part (ii) of the proposition follows by noting that SLE can remove l_{N-1} due to l, resulting in the formula

$$pre(\Phi_N) \;:=\; \bigcup_{i=1}^{2N-2} \mathbf{P}_i \cup \bigcup_{i=1}^{N-2} \mathbf{H}_i \cup$$
$$\left\{ \left(\bigvee_{j=1}^{N} p_k^j \right)^{\{l\}} \;\middle|\; k = (N-1)..(2N-1) \right\}.$$

The only LMUS of the preprocessed formula is $\{l\}$, which is why both algorithms are guaranteed to need only a single unsatisfiable and a single satisfiable SAT-solver call, and furthermore, why CG-MaxSAT needs no clause replication steps, during solving. □

6 EXPERIMENTS

Complementing the theoretical analysis, we evaluate the practical effects of SLE on the 2015 MaxSAT Evaluation benchmarks (http://www.maxsat.udl.cat/15/). We observe that SLE is beneficial especially on *industrial* weighted partial benchmark instances. When applying SLE in conjunction with the LCNF-lifted SAT-based preprocessing techniques (LVE, LSSR, LSE, LBCE), noticeably more labels can be removed than without applying SLE. Furthermore, SLE improves the overall performance of various state-of-the-art MaxSAT solvers on industrial weighted partial benchmarks.

All reported solving times include the time spent in preprocessing as well as in the actual MaxSAT solving. The experiments were run on 2.53-GHz Intel Xeon quad-core machines with 32-GB RAM under Linux. A per-instance timeout of 1800 seconds and a memory limit of 30 GB were enforced.

We implemented SLE by extending the Coprocessor 2.0 SAT preprocessor [34] in the following way. Given a MaxSAT instance as input, we convert the instance to LCNF, apply Coprocessor to preprocess the LCNF, and then convert the preprocessed LCNF back to a MaxSAT instance. LVE, LSSR LSE, LSSR, and LBCE are realized by representing a labelled clause C^L as $C \vee \bigvee_{l_i \in L} a_i$ in Coprocessor, applying the existing implementations of VE, SSR, SE and BCE, while forbidding the elimination of any of the a_i variables corresponding to the labels.

A simple way of implementing SLE consists of explicitly checking for each label l whether or not l is subsumed. A potentially more efficient way of implementing SLE would be to track the resolvents produced by LVE and only check labels that have appeared in resolvents produced. However, as shown in Figure 3, even the simple implementation appears to be sufficient; we did not observe any significant increase in total preprocessing time (w/pre+SLE) compared to not using SLE (w/pre). We also note that SLE does not increase overall memory consumption wrt SAT-based preprocessing.

The fraction of labels (i.e. soft clauses) remaining after preprocessing with and without SLE (applying in both cases LVE, LSSR, LSE, and LBCE) is shown in Figure 2 for both unweighted and weighted partial industrial and crafted instances. SLE is effective in removing additional labels in particular on the industrial weighted partial instances. For example, for one third of the instances ($x = 0.3$), with SLE close to 80% of the labels are eliminated ($y \approx 0.2$, i.e., some 20% of the labels remain afterwards); in comparison, without SLE only $\approx 45\%$ are eliminated. As a side-note, when examining the instance families in more detail, we found that out of the 172 industrial benchmarks in which no labels were removable by preprocessing, 151 were new instances in the 2015 evaluation. In fact, when preprocessing the 2014 evaluation instances—which are a subset of the 2015 evaluation instances—using SLE, at least 80% of the labels are eliminated from over 50% of the instances. This suggests that, in terms of SLE, the instances added for 2015 are structurally different from the ones from 2014.

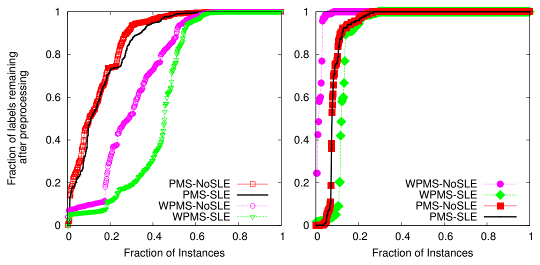

Figure 2: Fraction of labels remaining in industrial (left) and crafted (right) unweighted (PMS) and weighted (WPMS) benchmarks after preprocessing with and without SLE.

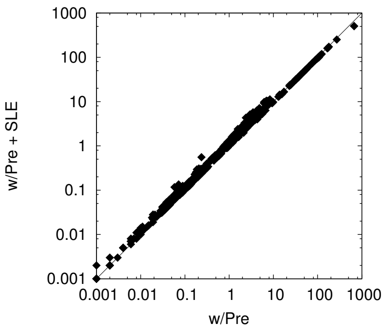

Figure 3: Influence of SLE on preprocessing time

Table 1: Number of solved industrial weighted partial benchmarks and total time spent on solved instances without preprocessing (default), with SAT-based preprocessing (w/pre), and with both SAT-based preprocessing and SLE (w/pre+SLE).

	Solved instances (total running time over solved in seconds)			
config.	**Eva**	**LMHS**	**Open-WBO**	**Primal-Dual**
default	379 (22,543)	354 (50,981)	331 (15,762)	390 (18,423)
w/pre	384 (20,613)	368 (46,525)	369 (12,345)	391 (15,267)
w/pre+SLE	**386 (19,138)**	**389** (48,277)	**369 (11,739)**	**392 (13,925)**

The additional simplifications obtained via SLE are also reflected in the total number of solved instances and solver runtimes on industrial weighted partial instances. Results are shown in Table 1 for the state-of-the-art MaxSAT solvers Eva [40], core-guided, best industrial weighted partial solver in 2014; LMHS [11], one of the best crafted and industrial weighted partial solvers in 2015, a labelled lifting of the SAT-IP hybrid MaxSAT solver MaxHS [18]; Open-WBO [36], one of the best industrial unweighted solvers in 2015; and Primal-Dual [12], a new core-guided solver from 2015. SAT-based preprocessing together with SLE results in the highest number of solved instances for each of the solvers. The increase in the number of solved instances is especially noticeable for LMHS. SLE also decreases the total runtime over all solved instances for each of the solvers. For example, for both Eva and Primal-Dual, using SLE improves further on applying only SAT-based preprocessing by decreasing the total runtime by approximately 10%, at the same time enabling Primal-Dual and Eva to solve one and two more instances, respectively. Finally, Figure 4 shows a comparison the running times of the individual instances with the solvers are presented in the order LMHS (first column), Eva (second), Open-WBO (third), and Primal-Dual (fourth column). For each solver, we compare runtimes on logscale when applying SLE together with LVE, LSSR, LSE, and LBCE ('w/pre+SLE') to (i) without preprocessing (left), and (ii) preprocessing only with LVE, LSSR, LSE, and LBCE ('w/pre", right). For a majority of the instances, SLE improves the total solving time of each of the solvers both compared to using no preprocessing, and only using LVE, LSSR, LSE and LBCE.

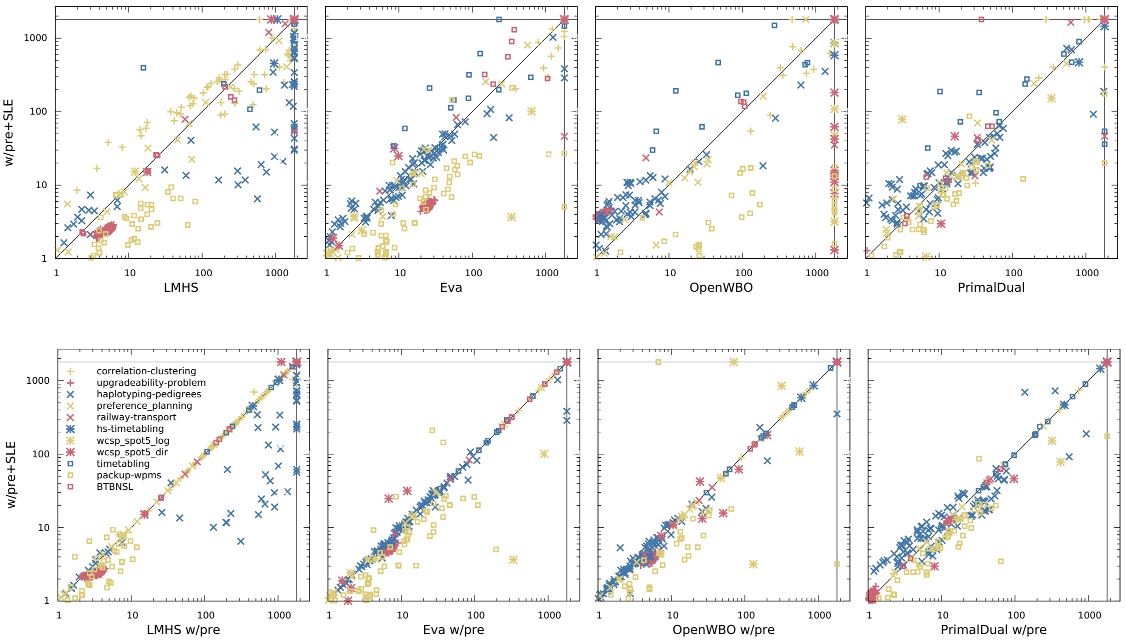

Figure 4: Effect of SLE on runtimes without (top) and with (bottom) other preprocessing on industrial weighted partial instances.

7 CONCLUSIONS

We proposed *subsumed label elimination* (SLE) as a MaxSAT preprocessing technique that is beneficial to apply in conjunction with SAT-based preprocessing techniques before MaxSAT solving. SLE is orthogonal to SAT-based preprocessing in that SLE can eliminate redundant auxiliary variables (labels) from clauses irrespective of the original variables occurring in clauses. On the level of labelled CNFs, this accounts to removing redundant labels from LMUSes, thereby resulting in cases in a decrease in the number and sizes of MUSes of MaxSAT instances. Furthermore, SLE has the potential to drastically reduce the number of iterations performed by core-guided MaxSAT solvers, currently one of the important classes of MaxSAT solvers. Applying SLE further improves the running times of various state-of-the-art MaxSAT solvers on standard industrial weighted partial benchmarks. For future work, we aim to study more general notions of redundancies over labels in LCNFs to obtain further label-based preprocessing techniques for MaxSAT, as well as to study potential applications in MUS extraction.

ACKNOWLEDGEMENTS

This work has been funded by Academy of Finland, grants 251170 COIN, 276412, and 284591; and Doctoral School in Computer Science DoCS and Research Funds of the University of Helsinki.

A SLE and Dominance in Binate Covering

SLE (for MaxSAT) can be viewed as the counterpart of the so-called *dominance rule* proposed in the early 90s in conjunction with branch-and-bound approaches for the so-called *binate covering problem* [17, 16] with applications in logic synthesis. In short, in the binate covering problem, we are given a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ over the variables $x_1,\ldots,x_n$, and a function $cost\colon \{1..n\} \to \mathbb{N}$ assigning a non-negative cost $cost(i)$ to each variable x_i. The task is to find a truth assignment τ over $x_1,\ldots,x_n$ that minimizes $\sum_{i=1}^{n} \tau(x_i) \cdot cost(i)$ subject to $f(\tau(x_1),\ldots,\tau(x_n)) = 1$. The dominance rule for binate covering is described in [17] for the so-called modified covering matrix representation of binate covering for Boolean functions in CNF. We interpret the rule directly on the definition as follows: variable x_i dominates x_j if (i) the literal x_i occurs in a clause C whenever the literal x_j occurs in C; (ii) $\neg x_j$ occurs in a clause C whenever $\neg x_i$ occurs in C; and (iii) $cost(x_i) \leq cost(x_j)$. A dominated variable can be assigned to 0.

A LCNF-MaxSAT instance (Φ, w) can be viewed as an instance of binate covering by viewing each labelled clause $C^L \in \Phi$ as the clause $C \vee L$, and letting $cost(l) = w(l)$ for each $l \in Lbls(\Phi)$ and $cost(x) = 0$ for each variable in $\bigcup Cl(\Phi)$. After this reduction, one can observe that, for any label $l \in Lbls(\Phi)$, it holds that l is dominated in the resulting binate covering instance if and only if SLE can eliminate l from (Φ, w).

REFERENCES

[1] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy, 'Solving (weighted) partial MaxSAT through satisfiability testing', in *Proc. SAT*, volume 5584 of *Lecture Notes in Computer Science*, pp. 427–440. Springer, (2009).

[2] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy, 'SAT-based MaxSAT algorithms', *Artificial Intelligence*, **196**, 77–105, (2013).

[3] Carlos Ansótegui, Idelfonso Izquierdo, Felip Manyà, and José Torres-Jiménez, 'A Max-SAT-based approach to constructing optimal covering arrays', in *Proc. CCIA*, volume 256 of *Frontiers in Artificial Intelligence and Applications*, pp. 51–59. IOS Press, (2013).

[4] Josep Argelich, Daniel Le Berre, Inês Lynce, João P. Marques-Silva, and Pascal Rapicault, 'Solving linux upgradeability problems using boolean optimization', in *Proc. LoCoCo*, volume 29 of *EPTCS*, pp. 11–22, (2010).

[5] Josep Argelich, Chu Min Li, and Felip Manyà, 'A preprocessor for Max-SAT solvers', in *Proc. SAT*, volume 4996 of *Lecture Notes in Computer Science*, pp. 15–20. Springer, (2008).

[6] Anton Belov and Joao Marques-Silva, 'Generalizing redundancy in propositional logic: Foundations and hitting sets duality', *CoRR*, **abs/1207.1257**, (2012).

[7] Anton Belov, Antonio Morgado, and Joao Marques-Silva, 'SAT-based preprocessing for MaxSAT', in *Proc. LPAR-19*, volume 8312 of *Lecture Notes in Computer Science*, pp. 96–111. Springer, (2013).

[8] Jeremias Berg and Matti Järvisalo, 'SAT-based approaches to treewidth computation: An evaluation', in *Proc. ICTAI*, pp. 328–335. IEEE Computer Society, (2014).

[9] Jeremias Berg and Matti Järvisalo, 'Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability', *Artificial Intelligence*, (2015). in press.

[10] Jeremias Berg, Matti Järvisalo, and Brandon Malone, 'Learning optimal bounded treewidth Bayesian networks via maximum satisfiability', in *Proc. AISTATS*, volume 33, pp. 86–95. JMLR, (2014).

[11] Jeremias Berg, Paul Saikko, and Matti Järvisalo, 'Improving the effectiveness of SAT-based preprocessing for MaxSAT', in *Proc. IJCAI*, pp. 239–245. AAAI Press, (2015).

[12] Nikolaj Bjørner and Nina Narodytska, 'Maximum satisfiability using cores and correction sets', in *Proc. IJCAI*, pp. 246–252. AAAI Press, (2015).

[13] Maria Luisa Bonet, Jordi Levy, and Felip Manyà, 'Resolution for Max-SAT', *Artificial Intelligence*, **171**(8-9), 606–618, (2007).

[14] Yibin Chen, Sean Safarpour, João Marques-Silva, and Andreas G. Veneris, 'Automated design debugging with maximum satisfiability', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **29**(11), 1804–1817, (2010).

[15] Yibin Chen, Sean Safarpour, Andreas G. Veneris, and João P. Marques-Silva, 'Spatial and temporal design debug using partial MaxSAT', in *Proc. 19th ACM Great Lakes Symposium on VLSI*, pp. 345–350. ACM, (2009).

[16] Olivier Coudert, 'On solving covering problems', in *Proc. DAC*, pp. 197–202. ACM Press, (1996).

[17] Olivier Coudert and Jean Christophe Madre, 'New ideas for solving covering problems', in *Proc. DAC*, pp. 641–646. ACM Press, (1995).

[18] Jessica Davies and Fahiem Bacchus, 'Exploiting the power of MIP solvers in MaxSAT', in *Proc. SAT*, volume 7962 of *Lecture Notes in Computer Science*, pp. 166–181. Springer, (2013).

[19] Jessica Davies and Fahiem Bacchus, 'Postponing optimization to speed up MAXSAT solving', in *Proc. CP*, volume 8124 of *Lecture Notes in Computer Science*, pp. 247–262. Springer, (2013).

[20] Niklas Eén and Armin Biere, 'Effective preprocessing in SAT through variable and clause elimination', in *Proc. SAT*, volume 3569 of *Lecture Notes in Computer Science*, pp. 61–75. Springer, (2005).

[21] Zhiwen Fang, Chu-Min Li, Kan Qiao, Xu Feng, and Ke Xu, 'Solving maximum weight clique using maximum satisfiability reasoning', in *Proc. ECAI*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 303–308. IOS Press, (2014).

[22] Zhaohui Fu and Sharad Malik, 'On solving the partial MaxSAT problem', in *Proc. SAT*, volume 4121 of *Lecture Notes in Computer Science*, pp. 252–265. Springer, (2006).

[23] Joao Guerra and Ines Lynce, 'Reasoning over biological networks using maximum satisfiability', in *Proc. CP*, volume 7514 of *Lecture Notes in Computer Science*, pp. 941–956. Springer, (2012).

[24] Armin Haken, 'The intractability of resolution', *Theoretical Computer Science*, **39**, 297–308, (1985).

[25] Federico Heras, Antonio Morgado, and Joao Marques-Silva, 'Core-guided binary search algorithms for maximum satisfiability', in *Proc. AAAI*. AAAI Press, (2011).

[26] Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere, 'Clause elimination for SAT and QSAT', *Journal of Artificial Intelligence Research*, **53**, 127–168, (2015).

[27] Alexey Ignatiev, Mikolás Janota, and João Marques-Silva, 'Towards efficient optimization in package management systems', in *Proc. ICSE*, pp. 745–755. ACM, (2014).

[28] Matti Järvisalo, Armin Biere, and Marijn Heule, 'Blocked clause elimination', in *Proc. TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pp. 129–144. Springer, (2010).

[29] Matti Järvisalo, Marijn Heule, and Armin Biere, 'Inprocessing rules', in *Proc. IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pp. 355–370. Springer, (2012).

[30] Manu Jose and Rupak Majumdar, 'Cause clue clauses: error localization using maximum satisfiability', in *Proc. PLDI*, pp. 437–446. ACM, (2011).

[31] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes, 'Exploiting cycle structures in Max-SAT', in *Proc. SAT*, volume 5584 of *Lecture Notes in Computer Science*, pp. 467–480. Springer, (2009).

[32] Inês Lynce and João Marques-Silva, 'Restoring CSP satisfiability with MaxSAT', *Fundam. Inform.*, **107**(2-3), 249–266, (2011).

[33] Vasco M. Manquinho, João P. Marques-Silva, and Jordi Planes, 'Algorithms for weighted boolean optimization', in *Proc. SAT*, volume 5584 of *Lecture Notes in Computer Science*, pp. 495–508. Springer, (2009).

[34] Norbert Manthey, 'Coprocessor 2.0 - A flexible CNF simplifier', in *Proc. SAT*, volume 7317 of *Lecture Notes in Computer Science*, pp. 436–441. Springer, (2012).

[35] Joao Marques-Silva, Mikolas Janota, Alexey Ignatiev, and Antonio Morgado, 'Efficient model based diagnosis with maximum satisfiability', in *Proc. IJCAI*, pp. 1966–1972. AAAI Press, (2015).

[36] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Ines Lynce, 'Incremental cardinality constraints for MaxSAT', in *Proc. CP*, volume 8656 of *Lecture Notes in Computer Science*, pp. 531–548. Springer, (2014).

[37] Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva, 'Core-guided MaxSAT with soft cardinality constraints', in *Proc. CP*, volume 8656 of *Lecture Notes in Computer Science*, pp. 564–573. Springer, (2014).

[38] Antonio Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and Joao Marques-Silva, 'Iterative and core-guided MaxSAT solving: A survey and assessment', *Constraints*, **18**(4), 478–534, (2013).

[39] António Morgado, Mark H. Liffiton, and João Marques-Silva, 'MaxSAT-based MCS enumeration', in *Revised Selected Papers of HVC 2012*, volume 7857 of *Lecture Notes in Computer Science*, pp. 86–101. Springer, (2013).

[40] Nina Narodytska and Fahiem Bacchus, 'Maximum satisfiability using core-guided MaxSAT resolution', in *Proc. AAAI*, pp. 2717–2723. AAAI Press, (2014).

[41] James D. Park, 'Using weighted MAX-SAT engines to solve MPE', in *Proc. AAAI*, pp. 682–687. AAAI Press / The MIT Press, (2002).

[42] Johannes Peter Wallner, Andreas Niskanen, and Matti Järvisalo, 'Complexity results and algorithms for extension enforcement in abstract argumentation', in *Proc. AAAI*, pp. 1088–1094. AAAI Press, (2016).

[43] Lei Zhang and Fahiem Bacchus, 'MAXSAT heuristics for cost optimal planning', in *Proc. AAAI*. AAAI Press, (2012).

[44] Charlie S. Zhu, Georg Weissenbacher, and Sharad Malik, 'Post-silicon fault localisation using maximum satisfiability and backbones', in *Proc. FMCAD*, pp. 63–66. FMCAD Inc., (2011).

Paper V

Jeremias Berg and Matti Järvisalo

Cost-Optimal Constrained Correlation Clustering via Weighted Partial Maximum Satisfiability

V

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Cost-optimal constrained correlation clustering via weighted partial Maximum Satisfiability [☆]

Jeremias Berg, Matti Järvisalo [*]

Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Finland

A R T I C L E   I N F O

Article history:
Received in revised form 18 June 2015
Accepted 3 July 2015
Available online 9 July 2015

Keywords:
Boolean optimization
Boolean satisfiability
Maximum satisfiability
Correlation clustering
Cost-optimal clustering
Constrained clustering

A B S T R A C T

Integration of the fields of constraint solving and data mining and machine learning has recently been identified within the AI community as an important research direction with high potential. This work contributes to this direction by providing a first study on the applicability of state-of-the-art Boolean optimization procedures to cost-optimal correlation clustering under constraints in a general similarity-based setting. We develop exact formulations of the correlation clustering task as Maximum Satisfiability (MaxSAT), the optimization version of the Boolean satisfiability (SAT) problem. For obtaining cost-optimal clusterings, we apply a state-of-the-art MaxSAT solver for solving the resulting MaxSAT instances optimally, resulting in cost-optimal clusterings. We experimentally evaluate the MaxSAT-based approaches to cost-optimal correlation clustering, both on the scalability of our method and the quality of the clusterings obtained. Furthermore, we show how the approach extends to *constrained* correlation clustering, where additional user knowledge is imposed as constraints on the optimal clusterings of interest. We show experimentally that added user knowledge allows clustering larger datasets, and at the same time tends to decrease the running time of our approach. We also investigate the effects of MaxSAT-level preprocessing, symmetry breaking, and the choice of the MaxSAT solver on the efficiency of the approach.

1. Introduction

Integration of the fields of constraint solving and data mining and machine learning has recently been identified within the AI community as an important research direction with high potential. This work contributes to this direction by studying the applicability of Boolean optimization to cost-optimal correlation clustering under constraints.

A common problem setting in data analysis is a set of data points together with some information regarding their pairwise similarities from which some interesting underlying structure needs to be discovered. One way of approaching the

[☆] This work is supported by Academy of Finland (grants 276412, 284591, and 251170 Finnish Centre of Excellence in Computational Inference Research COIN), Doctoral Program in Computer Science DOCS, Research Funds of the University of Helsinki, and Finnish Funding Agency for Technology and Innovation (project D2I: From Data to Intelligence). The authors thank Jessica Davies for providing the MaxHS solver version used in the experiments. A preliminary version of this work appeared as [1] and was presented at the 2013 ICDM workshops. This article thoroughly revises and extends the earlier workshop paper considerable, for example by addressing the problem in a more general weighted setting, by introducing a third improved MaxSAT encoding, by extended experiments including comparisons with quadratic integer programming and several approximative algorithms, application of SAT-based preprocessing, symmetry breaking, and a MaxSAT solver comparison, as well as inclusion of full formal proofs and extended background and discussions.

[*] Corresponding author. Tel.: +358 50 3199 248; fax: +358 9 1915 1120.
E-mail addresses: jeremias.berg@cs.helsinki.fi (J. Berg), matti.jarvisalo@cs.helsinki.fi (M. Järvisalo).

http://dx.doi.org/10.1016/j.artint.2015.07.001
0004-3702/© 2015 Elsevier B.V. All rights reserved.

problem is to attempt to divide the data into subgroups in a meaningful way, for example, so that data points in the same group are more similar to each other than to data points in other groups [2]. Discovering an optimal way of making such a division is in most settings computationally challenging and an active area of research [3]. A general term for problems of this kind is *clustering*: the groups the data is partitioned into are called *clusters*, and a partitioning of the dataset is called a *clustering* of the data.

In this work, we study the *correlation clustering* paradigm [4] in a general similarity-based setting. Correlation clustering is a well-studied [5–9] NP-hard problem. Given a labeled undirected graph with each edge labeled with either a positive or a negative label, the objective of correlation clustering is to cluster the nodes of the graph in a way which minimizes the number of positive edges between different clusters and negative edges within clusters. Taking a more general view to correlation clustering, we study the problem setting of *weighted* correlation clustering, in which each edge is associated with a weight (instead of merely a negative or positive label), indicating our confidence in that label. In the more general weighted case, the objective of correlation clustering is to minimize the sum of the weights of the positive edges between different clusters and the negative edges within clusters.

The correlation clustering paradigm is geared towards classifying data based on qualitative similarity information—as opposed to quantitative information—of pairs of data points. In contrast to other typical clustering paradigms, correlation clustering does not require the number of clusters as input. This makes it especially well-suited for settings in which the true number of clusters is unknown—which is often the case when dealing with real-world data. As a concrete example, consider the problem of clustering documents by topic without any prior knowledge on what those topics might be, based only on similarity information (edges) between pairs of different documents [4,10]. Indeed, correlation clustering has various applications in biosciences [11], social network analysis and information retrieval [12–14]. Furthermore, the related problem of *consensus clustering* [15], with recent applications in bioinformatics and in particular microarray data analysis [16–19], can also be naturally cast as correlation clustering.

Due to NP-hardness of correlation clustering [4], most algorithmic work on the problem has been heuristic, focusing on local search and approximative algorithms. While strong approximation algorithms have been proposed [4–6,9]—providing up to constant-factor approximations in restricted settings—these algorithms are unable to provide actual cost-optimal solutions in general. In this work, we take a different approach: we study the applicability of state-of-the-art Boolean optimization techniques to *cost-optimally* solving real-world instances of the correlation clustering problem. A baseline motivation for this work are the recent advances in applying constraint programming for developing generic approaches to common data analysis problems [20–25]. In a constraint programming based approach, the data analysis problem is stated in a declarative fashion within some constraint language, and then a generic solver for that language is used for solving the resulting instance.

Harnessing constraint solving for data analysis tasks has two key motivations. Firstly, declarative optimization systems allow for finding provably cost-optimal solutions. While heuristic approaches allow for scaling to very large datasets, quickly obtaining some hopefully meaningful clustering, the provably cost-optimal solutions obtained by the declarative approach can result in notably better clusterings which provide better insights into the data. This can be valuable especially when working on smaller scientific datasets which have taken years to collect [26]. Secondly, the declarative approach allows for easily integrating various types of additional constraints over the solution space at hand. This way, a user (domain data expert) may specify properties of solutions that are of interest to the user, without needing to extend available specialized algorithms in a non-trivial way to cope with such additional constraints. A constraint-based framework for clustering problems is well-suited for problem instances where some form of domain specific knowledge might be required in order to obtain meaningful clusterings. The paradigm for clustering problems of this type is known as *constrained clustering* [27–29]. Recently, Boolean satisfiability (SAT) [30] based approaches to solving constrained clustering within other clustering problems have been proposed [23,31], However, to the best of our knowledge the only work done on constrained correlation clustering is the linear programming based approach of [10]; this work is the first study on the applicability of Maximum Satisfiability (MaxSAT) [32], a well-known optimization version of SAT, to correlation clustering under constraints. The problem definition we study covers correlation clustering with additional constraints that, e.g., either force or forbid a pair of points from being assigned to the same cluster; known as must-link and cannot-link constraints [27].

1.1. Contributions

We present a novel and extensible MaxSAT-based approach to optimal correlation clustering. Using propositional logic as the declarative language, we formulate the correlation clustering task in an exact fashion as weighted partial MaxSAT [32] and apply a state-of-the-art MaxSAT solver to solve the resulting MaxSAT instance optimally. To our best knowledge this is the first practical approach to *exactly* solving correlation clustering for finding *cost-optimal* clusterings, i.e., optimal clusterings w.r.t. the actual objective function of the problem, for real-world datasets with hundreds of elements. In contrast, most of the previous work on correlation clustering has mainly focused on approximation algorithms and greedy local-search techniques which cannot in general find optimal clusterings.

At the core of the approach, we present three different MaxSAT formulations of correlation clustering, and provide formal proofs for their correctness. We experimentally evaluate our approach on real-world datasets and compare the approach to both two alternative exact approaches, based on linear and quadratic integer programming [5,33], and two approximation algorithms [5,34]. The results show that our approach can provide cost-optimal solutions and scales better than competing exact integer and quadratic programming formulations. Furthermore, our approach performs especially well in terms of solution cost on sparse datasets (with many missing similarity entries), outperforming approximative methods even when the approximative methods are given full similarity information. Our approach easily extends to the task of *constrained correlation clustering*, which allows for the user to specify the clusterings of one's interest by imposing hard user-defined constraints over the search space of clusterings. We explain how different types of constraints can be handled within a MaxSAT-based approach to cost-optimally solving constrained correlation clustering instances. While approaches to constrained clustering have been proposed previously for different clustering paradigms [27–29,35,36,23], the only previous work on constrained correlation clustering that we know of is [10]. However, their approach is approximative and the experiments are done on smaller datasets. In contrast, we show experimentally that added user knowledge allows clustering on larger datasets as it tends to notably decrease the running time of the approach. We also provide experimental results on MaxSAT-specific aspects of solving the correlation clustering instances, considering the effects of MaxSAT-level preprocessing, symmetry breaking, as well as the choice of the MaxSAT solver used on the efficiency of the approach.

1.2. Paper organization

In Section 2 we provide a generic problem definition for correlation clustering that is used throughout this article. Our definition covers both correlation clustering and constrained correlation clustering. We also demonstrate how a symmetric similarity measure simplifies the objective function of the clustering problem and show that a similarity measure can always be assumed to be symmetric. In Sections 3 and 4 we overview previously proposed linear and quadratic integer programs for solving correlation clustering exactly. In Section 5 we provide necessary background on Maximum Satisfiability. The MaxSAT encodings of correlation clustering are detailed in Sections 6, 7 and 8, respectively. Extensive experimental results are provided in Section 9. Finally, we present a short survey on related work in Section 10 and give some concluding remarks in Section 11. Formal proofs of the theorems presented in the paper are given in Appendix A.

2. Problem setting

In this section, we present the general similarity-based problem setting under which we study correlation clustering in both unconstrained and constrained settings.

2.1. Problem definition

Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty, -\infty\}$, $V = \{v_1, \ldots, v_N\}$ a set of N data points that we wish to cluster, and $W \in \overline{\mathbb{R}}^{N \times N}$ a *similarity matrix*. We denote the element on row i column j in W by $W(i, j)$. This input can be viewed as a weighted graph, as demonstrated by the following example.

Example 1. Let $V = \{v_1, v_2, v_3, v_4\}$ be a set of data points and consider the similarity matrix W given in Fig. 1 on the left. We can view this input as a directed graph $G = (V, E)$ where $(v_i, v_j) \in E$ if $W(i, j) \neq 0$, and the weight of each edge (v_i, v_j) is equal to $W(i, j)$. Fig. 1(right) illustrates the graph corresponding to W. In case the similarity matrix is symmetric, i.e., $W(i, j) = W(j, i)$ for all i and j, the graph underlying W is essentially undirected.

$$W = \begin{bmatrix} \infty & 0 & -4.5 & 40 \\ -0.3 & \infty & -\infty & 65 \\ 2.5 & 0 & \infty & 0 \\ 0 & 4.5 & 0 & \infty \end{bmatrix}$$

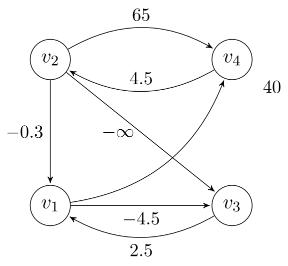

Fig. 1. An example similarity matrix and its graph presentation.

The intuition behind the similarity matrix is that it expresses preferences on whether or not two points v_i and v_j should be assigned to the same cluster; a positive value indicates that v_i and v_j should be assigned to the same cluster, a negative value that they should not. We say that points v_i and v_j are *similar* if $W(i, j) \in \mathbb{R}$ and $W(i, j) > 0$. If $W(i, j) < 0$ and $W(i, j) \in \mathbb{R}$, we say that v_i and v_j are *dissimilar*. In the most general setting, neither the requirement of assigning pairs of points to the same (different) cluster(s) nor the notion of pairs of points being (dis)similar are required to be symmetric relations.

Any function $cl: V \to \mathbb{N}$ is a solution to the clustering problem, representing a clustering of the data points into clusters indexed with natural numbers. We say that two points v_i and v_j are *co-clustered* if $cl(v_i) = cl(v_j)$. Note that our formulation

Input: A set of N data points $V = \{v_1 \dots v_N\}$ and a symmetric similarity matrix $W \in \overline{\mathbb{R}}^{N \times N}$.

Output: A function $cl^*\colon V \to \mathbb{N}$ such that $cl^* \in \operatorname{argmin}_{cl::V \to \mathbb{N}}(H(W, cl))$.

Fig. 2. The constrained correlation clustering problem.

allows forcing two points to the same or different clusters. If $W(i, j) = \infty$ for some i and j, then v_i and v_j have to be co-clustered. Analogously, if $W(i, j) = -\infty$, then v_i and v_j are not allowed to be co-clustered. If the infinite values are in conflict with each other, the problem instance is infeasible. These additional hard constraints are commonly referred to as *must-link* (ML) and *cannot-link* (CL) constraints [27]. We will use the following definition to incorporate the intended semantics of the infinite values onto the possible clusterings. Given a similarity matrix W, we say that a clustering cl *respects the infinite values of* W, if $cl(v_i) = cl(v_j)$ whenever $W(i, j) = \infty$ and $cl(v_i) \neq cl(v_j)$ whenever $W(i, j) = -\infty$.

Given a cost function G such that $G(W, cl) \in \mathbb{R}$ for every solution cl, we say that a clustering cl (of V) is *optimal* under W as measured by G, if cl respects the infinite values of W and $G(W, cl) \leq G(W, cl')$ holds for any clustering cl' (of V) that respects the infinite values of W. The definition is sufficient for all purposes as we can always turn a function G we wish to maximize into a minimization problem by considering the function $-G$. For a given similarity matrix W we use $\operatorname{argmin}_{cl}(G(W, cl))$ to denote the set of optimal clusterings under W as measured by G.

In this work we focus on the cost function of *correlation clustering* with additional must-link and cannot-link constraints. In correlation clustering [4,9] we are given a pairwise similarity measure over a set of data points. The task is then to cluster the nodes in a way that maximizes the number of similar points co-clustered and minimizes the number of dissimilar points co-clustered. More formally, given a symmetric similarity matrix W, the task is to find a clustering which minimizes the cost function

$$H(W, cl) = \sum_{\substack{cl(v_i) = cl(v_j) \\ i<j}} (\mathcal{I}[-\infty < W(i, j) < 0] \cdot |W(i, j)|) + \sum_{\substack{cl(v_i) \neq cl(v_j) \\ i<j}} (\mathcal{I}[\infty > W(i, j) > 0] \cdot W(i, j)) \tag{1}$$

where $\mathcal{I}[b]$ is an indicator function which takes the value 1 if the condition b is true, else $I[b] = 0$. Fig. 2 gives a precise formulation of constrained correlation clustering used throughout this work.

This definition covers all variants of correlation clustering that we are aware of. For example, the definition of [4] where the input consists of a complete graph with each edge labeled by a $+$ or $-$ is equivalent to restricting the input similarity matrix to only contain values from $\{-1, 1\}$ and specifically not to contain infinite values. Furthermore, the assumption of symmetric input can be made without loss of generality, as detailed in Section 2.2.

Example 2. Let $V = \{v_1, v_2, v_3, v_4\}$ be a set of data points and consider the similarity matrix given in Fig. 3 on the left. Fig. 3(right) illustrates one possible solution cl to the correlation clustering problem for this input data. In described solution, $cl(v_1) = cl(v_2) = cl(v_3) \neq cl(v_4)$. The cost of cl is

$$\begin{aligned} H(W, cl) = {} & (\mathcal{I}[W(1,2) < 0] \cdot |W(1,2)| + \mathcal{I}[W(1,3) < 0] \cdot |W(1,3)| + \mathcal{I}[W(2,3) < 0] \cdot |W(2,3)|) + \\ & (\mathcal{I}[W(1,4) > 0] \cdot W(1,4) + \mathcal{I}[W(2,4) > 0] \cdot W(2,4) + \mathcal{I}[W(3,4) > 0] \cdot W(3,4)) \\ = {} & |W(1,2)| + W(3,4) = 3.3. \end{aligned}$$

$$W = \begin{bmatrix} \infty & -0.3 & 2.5 & 0 \\ -0.3 & \infty & 4 & -5 \\ 2.5 & 4 & \infty & 3 \\ 0 & -5 & 3 & \infty \end{bmatrix}$$

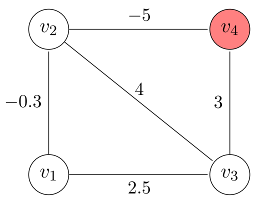

Fig. 3. An example similarity matrix and the graphical representation of a solution to the correlation clustering problem.

In contrast to many other clustering problems, deciding the number of clusters is in the most general case part of the correlation clustering problem. However, as every point is assigned to exactly one cluster, in practice it is enough to search over all functions $cl\colon V \to \{1, \dots, N\}$.

2.2. On the assumption of symmetric similarities

We will now show that the assumption of symmetric similarity matrices in our problem definition (Fig. 2) can be done without loss of generality. Correlation clustering is often defined with 2 positive weights w^+_{ij} and w^-_{ij} for each pair of data points v_i, v_j as the input [5]. The intuition behind these weights is that they give a separate measure for the costs of not assigning v_i and v_j to the same (w^+_{ij}) and to different (w^-_{ij}) cluster(s). A straightforward method of modeling this in terms of our clustering setting would be to use a cost function such as

$$H'(W, cl) = \sum_{cl(v_i)=cl(v_j)} (\mathcal{I}[-\infty < W(i,j) < 0] \cdot |W(i,j)|) + \sum_{cl(v_i)\neq cl(v_j)} (\mathcal{I}[\infty > W(i,j) > 0] \cdot W(i,j)), \tag{2}$$

and letting $W(i,j) = w^+_{ij}$ and $W(j,i) = -w^-_{ij}$ for all $i < j$. However, this turns out to be unnecessary.

Theorem 1. *Let $V = \{v_1 \ldots v_N\}$ be a set of data points and W an asymmetric similarity matrix over V. Assume that for all i and j, $W(i,j) = \infty$ implies $W(j,i) \neq -\infty$ (from which it also follows that $W(i,j) = -\infty$ implies $W(j,i) \neq \infty$). Then there is a symmetric similarity matrix W^S such that*

$$\text{argmin}_{cl}(H(W^S, cl)) = \text{argmin}_{cl}(H'(W, cl)).$$

We note that the assumption in the theorem is minor. The condition can be checked in polynomial time, and if it does not hold, there are no feasible solutions to the constrained problem. A detailed proof of Theorem 1 is provided in Appendix A. The simpler objective function H simplifies the exact declarative formulations considered in this work.

2.3. Constrained clustering

In the clustering domain, the concept of a constraint is fairly abstract and the exact types of constraints that are feasible depend on the particular domain. A typical categorization of different types of constraints are *instance-level* constraints and *cluster-level constraints* [37]. Cluster-level constraints [36] deal with relationships between clusters. Examples of cluster-level constraints include constraints which enforce a predefined lower bounds for the similarity (distance) over clusters or a predefined upper bound on the dissimilarity of points within the clusters, as well as constraints requiring that the clustering contains at most/exactly/at least some fixed number of clusters or that all clusters contain at least a certain number of data points. Instance-level constraints deal with relationships between points. Two very well known examples are the already discussed ML and CL constraints. ML and CL constraints are have been shown to be flexible in the sense that many different types of constraints can be expressed in terms of them [37].

2.4. Consensus clustering

As detailed in [15], another problem closely related to correlation clustering is *consensus clustering*. In consensus clustering we are given a set V of data points and K different *clusterings* of V. The task is then to find a single *consensus clustering* which agrees as well as possible with the input clusterings. Consensus clustering fits into our problem definition by the following construction. For each pair of points $v_i, v_j \in V$ let s_{ij} be the number of clusterings in which v_i and v_j are co clustered and $d_{ij} = K - s_{ij}$ be the number of clusterings in which they are not. Now construct a similarity matrix W by assigning $W(i,j) = s_{ij}$ and $W(j,i) = -d_{ij}$ for each pair of data points and apply Theorem 1 to obtain the equivalent (in terms of correlation clustering) symmetric similarity matrix. Then an optimal solution to the resulting correlation clustering problem corresponds to an optimal solution to the consensus clustering problem. Consensus clustering is indeed also NP-hard [38]. Recently the problem has received more attention due to applications in bioinformatics and in particular microarray data analysis [16–19].

3. Correlation clustering as integer linear programming

An exact integer linear programming (ILP) formulation of correlation clustering has been proposed in [5,10]. We will now restate this integer linear programming formulation in terms of our generic problem setting.

Given a set $V = \{v_1, \ldots, v_N\}$ of N data points and a symmetric similarity matrix W, the integer program involves binary variables $x_{ij} \in \{0,1\}$, where $1 \leq i < j \leq N$. The intended interpretation of these variables is that $x_{ij} = 1$ iff v_i and v_j are co-clustered in any clustering. We note that the variables are only required whenever $i < j$. However, for notational convenience, we use x_{ij} and x_{ji} to denote the same variable. Using these variables, the set of optimal solutions to the following integer linear program represents the set of optimal clusterings of V under W [5].

$$\begin{array}{lll} \textsc{Minimize} & \displaystyle\sum_{\substack{-\infty < W(i,j) < 0 \\ i<j}} \big(x_{ij} \cdot |W(i,j)|\big) - \sum_{\substack{\infty > W(i,j) > 0 \\ i<j}} \big(x_{ij} \cdot W(i,j)\big) & \\ \text{subject to:} & x_{ij} + x_{jk} \leq 1 + x_{ik} & \text{for all distinct } i, j, k \\ & x_{ij} = 1 & \text{for all } W(i,j) = \infty \\ & x_{ij} = 0 & \text{for all } W(i,j) = -\infty \\ & x_{ij} \in \{0,1\} & \text{for all } i, j. \end{array} \tag{3}$$

The purpose of the *transitivity constraint* $x_{ij} + x_{jk} \leq 1 + x_{ik}$ is to ensure a *well-defined* clustering; for any $(v_i, v_j, v_k) \in V \times V \times V$, each of the points v_i, v_j, v_k must belong to exactly one cluster, and hence it follows that if points v_i, v_j are assigned to the same cluster and points v_j, v_k are assigned to the same cluster, by transitivity then points v_i, v_k should also be assigned to the same cluster. Stated as a linear constraint we require that if $x_{ij} + x_{jk} = 2$ then $x_{ik} = 1$, which is exactly what the transitivity constraint in the integer program demands. The purpose of the two other constraints is to ensure that the solution clustering respects the infinite values of W. Whenever $W(i, j) = \infty$, v_i and v_j have to be co-clustered, which in terms of the integer program is equivalent to $x_{ij} = 1$. Analogously, $W(i, j) = -\infty$ is equivalent to $x_{ij} = 0$. This formulation consists of $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N^3)$ constraints. In terms of practical considerations, this suggests poor scalability for larger datasets.

4. Correlation clustering as quadratic integer programming

A quadratic integer programming formulation of correlation clustering was proposed in [33]. In addition to the number of data points N, the quadratic integer programming (QIP) formulation requires one additional parameter K, an upper limit for the number of clusters that the solution clustering should contain. The formulation allows $K = N$ in which case the set of possible solutions to the quadratic program exactly matches the set of possible solutions to the integer linear programming formulation of correlation clustering and our general definition of correlation clustering (recall Fig. 2). We next restate the quadratic program in terms of our generic problem setting and the parameter K.

Given a set $V = \{v_1, \dots, v_N\}$ of N data points, an upper bound on the number of clusters K, and a symmetric similarity matrix W, the quadratic program involves binary variables $y_i^k \in \{0, 1\}$, where $1 \leq i \leq N$ and $1 \leq k \leq K$. The intended interpretation of the variables is that $y_i^k = 1$ iff data point v_i is assigned to cluster k. Using these variables, the set of optimal solutions to the following quadratic integer program represents the set of optimal clusterings of V under W [33].

$$\begin{array}{lll}
\textsc{Minimize} & \displaystyle\sum_{\substack{-\infty < W(i,j) < 0 \\ i<j}} \left(\sum_{k=1}^{K} \left(y_i^k y_j^k \right) \cdot |W(i,j)| \right) - \sum_{\substack{\infty > W(i,j) > 0 \\ i<j}} \left(\sum_{k=1}^{K} \left(y_i^k y_j^k \right) \cdot W(i,j) \right) & \\
\text{subject to:} & \displaystyle\sum_{k=1}^{K} y_i^k = 1 \qquad \text{for all } i & \\
& \displaystyle\sum_{k=1}^{K} \left(y_i^k y_j^k \right) = 1 \qquad \text{for all } W(i,j) = \infty & (4) \\
& \displaystyle\sum_{k=1}^{K} \left(y_i^k y_j^k \right) = 0 \qquad \text{for all } W(i,j) = -\infty & \\
& y_i^k \in \{0, 1\} \qquad \text{for all } i, k. &
\end{array}$$

For some intuition, note that the sum $\sum_{k=1}^{K} \left(y_i^k y_j^k \right)$ is equal to 1 only if points v_i and v_j are assigned to the same cluster in the solution clustering. The purpose of the $\sum_{k=1}^{K} y_i^k = 1$ for all i constraint is to ensure that the solution to the quadratic program corresponds to a well-defined clustering of the data. As all the variables used are binary, the constraint forces exactly one of the variables $y_i^1, \dots, y_i^K$ to 1 for all i, which in turn ensures that the corresponding data point v_i is assigned to exactly one cluster, as required for a well-defined clustering. This non-convex QIP consists of $\mathcal{O}(NK)$ variables and $\mathcal{O}(N + I)$ constraints where I is the number of infinite values in the input similarity matrix. We note that the non-convexity of the quadratic program can follow both from the integrality constraints as well as the similarity values themselves, as demonstrated by the following example.

Example 3. Consider the set $V = \{v_1, v_2, v_3\}$ of data points and the following similarity matrix over V:

$$W = \begin{bmatrix} \infty & -1 & -10 \\ -1 & \infty & 1 \\ -10 & 1 & \infty \end{bmatrix}.$$

For this similarity matrix and $K = N = 3$, the QIP in matrix form is

$$\begin{array}{ll}
\textsc{Minimize} & \frac{1}{2}(\mathbf{y})^T \mathbb{W}(\mathbf{y}) \\
\text{subject to:} & A\mathbf{y} = b \\
& \mathbf{y} \in \{0, 1\}^9,
\end{array}$$

where

$$\mathbf{y} = \begin{bmatrix} y_1^1 \\ y_1^2 \\ y_1^3 \\ y_2^1 \\ y_2^2 \\ y_2^3 \\ y_3^1 \\ y_3^2 \\ y_3^3 \end{bmatrix} \text{ and } \mathbb{W} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 10 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 10 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

with A and b chosen to fit the constraints $\sum_{k=1}^{K} y_i^k = 1$ for all i. For this instance of correlation clustering, the matrix $\mathbb{W}$ is indefinite. To see this, observe that it has both negative and positive eigenvalues, for example 10 and $-5 - 3\sqrt{3}$. Hence the objective function of the quadratic program in itself is not convex.

5. Maximum Satisfiability

Before describing our MaxSAT formulations of correlation clustering, we review necessary basic concepts related to Maximum Satisfiability.

5.1. Syntax and semantics

For a Boolean variable x, there are two literals, x and $\neg x$. A clause is a disjunction ($\vee$, logical OR) of literals and a truth assignment is a function from Boolean variables to $\{0, 1\}$. A clause C is satisfied by a truth assignment τ ($\tau(C) = 1$) if $\tau(x) = 1$ for a literal x in C, or $\tau(x) = 0$ for a literal $\neg x$ in C. A set F of clauses is satisfiable if there is an assignment τ satisfying all clauses in F ($\tau(F) = 1$), and unsatisfiable ($\tau(F) = 0$ for any assignment τ) otherwise.

An instance $F = (F_h, F_s, c)$ of the *weighted partial MaxSAT* problem consists of two sets of clauses, a set F_h of *hard* clauses and a set F_s of *soft* clauses, and a function $c : F_s \to \mathbb{R}^+$ that associates a non-negative cost with each of the soft clauses.[1] Any truth assignment τ that satisfies F_h is a *solution* to F. The *cost* $\textsc{cost}(F, \tau)$ of a solution τ to F is defined as

$$\textsc{cost}(F, \tau) = \sum_{C \in F_s} c(C) \cdot (1 - \tau(C)),$$

i.e., as the sum of the costs of the soft clauses not satisfied by τ. A solution τ is (globally) *optimal* for F if $\textsc{cost}(F, \tau) \leq \textsc{cost}(F, \tau')$ holds for any solution τ' to F. The cost of the optimal solutions of F is denoted by $\textsc{opt}(F)$. Given a weighted partial MaxSAT instance F, the weighted partial MaxSAT problem asks to find an optimal solution to F. For simplicity, we will from here on drop the term "weighted partial" when referring to weighted partial MaxSAT instances, and simply refer to them as MaxSAT instances.

Example 4. As an example of modeling problems with MaxSAT, consider the 3-coloring problem for the graph in Fig. 4. The coloring problem can be modeled with MaxSAT by forming a MaxSAT instance $F = (F_h, F_s, c)$ using a set of 15 boolean variables, $\{r_i, b_i, g_i \mid i = 1..5\}$. The intended semantics of a variable r_x is that the node x is colored red, similarly for g_x (green) and b_x (blue). The hard clauses in F restrict each node to be colored with exactly one color and the soft clauses represent the constraints forcing each pair of nodes sharing an edge to be colored with different colors. As clauses, this corresponds to

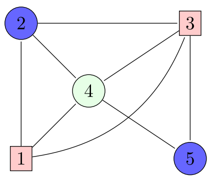

Fig. 4. An example graph together with one of its 3-colorings. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

[1] Our definition for the function c is more general than the standard $c : F_s \to \mathbb{N}^+$, which restricts the costs of soft clauses to be integral.

$$\begin{aligned}F_h = \{&(r_1 \vee b_1 \vee g_1), (r_2 \vee b_2 \vee g_2), (r_3 \vee b_3 \vee g_3), (r_4 \vee b_4 \vee g_4), (r_5 \vee b_5 \vee g_5),\\ &(\neg r_1 \vee \neg g_1), (\neg r_1 \vee \neg b_1), (\neg b_1 \vee \neg g_1), (\neg r_2 \vee \neg g_2), (\neg r_2 \vee \neg b_2), (\neg b_2 \vee \neg g_2),\\ &(\neg r_3 \vee \neg g_3), (\neg r_3 \vee \neg b_3), (\neg b_3 \vee \neg g_3), (\neg r_4 \vee \neg g_4), (\neg r_4 \vee \neg b_4), (\neg b_4 \vee \neg g_4),\\ &(\neg r_5 \vee \neg g_5), (\neg r_5 \vee \neg b_5), (\neg b_5 \vee \neg g_5)\}\end{aligned}$$

and

$$\begin{aligned}F_s = \{&(\neg r_1 \vee \neg r_2), (\neg b_1 \vee \neg b_2), (\neg g_1 \vee \neg g_2), (\neg r_1 \vee \neg r_3), (\neg b_1 \vee \neg b_3), (\neg g_1 \vee \neg g_3),\\ &(\neg r_1 \vee \neg r_4), (\neg b_1 \vee \neg b_4), (\neg g_1 \vee \neg g_4), (\neg r_2 \vee \neg r_3), (\neg b_2 \vee \neg b_3), (\neg g_2 \vee \neg g_3),\\ &(\neg r_2 \vee \neg r_4), (\neg b_2 \vee \neg b_4), (\neg g_2 \vee \neg g_4), (\neg r_3 \vee \neg r_4), (\neg b_3 \vee \neg b_4), (\neg g_3 \vee \neg g_4),\\ &(\neg r_3 \vee \neg r_5), (\neg b_3 \vee \neg b_5), (\neg g_3 \vee \neg g_5), (\neg r_4 \vee \neg r_5), (\neg b_4 \vee \neg b_5), (\neg g_4 \vee \neg g_5)\}\end{aligned}$$

with $c(w) = 1$ for all $w \in F_s$. An optimal solution τ to F is $\tau(r_1) = \tau(r_3) = \tau(b_5) = \tau(b_2) = \tau(g_4) = 1$ and $\tau(x) = 0$ for all other variables. The cost of this solution is 1, proving that any 3-coloring of the graph in Fig. 4 has to assign the same color to at least one pair of nodes sharing an edge.

5.2. Solving MaxSAT

Recent advances in MaxSAT solvers make MaxSAT a viable approach to finding globally (cost-)optimal solutions to various optimization problems with successful real-world applications such as hardware design debugging [39], post-silicon and C-code fault localization [40,41], reasoning over biological networks [42], and optimal Bayesian network structure learning [43]. As both SAT solvers and MaxSAT solvers continue improving, it is becoming commonly accepted that large problems can be solved in practice [44] and that the computational time is very much an empirical question and often not dominated by theoretical worst-case complexity. Indeed, MaxSAT is an active area of research [45–49]. We next provide a short overview of MaxSAT solvers. For a more comprehensive discussion, we refer the reader to [50,51].

Many of the state-of-the-art MaxSAT solvers aimed at efficiently solving real-world instances in practice make use of a SAT solver as a subroutine. By *relaxing* the soft clauses in the input formula, the MaxSAT solver can linearly search for the optimal solution to the instance by querying the SAT solver for the existence of a truth assignment (not) satisfying at least (at most) k soft clauses for different values of k. Intuitively, k can then either be an upper [46] or a lower bound [52,53] for the optimal solution. Another often used search strategy is binary search [49,47]. This basic idea of the algorithm has been improved by exploiting the fact that whenever invoked on an unsatisfiable set of clauses, a modern SAT-solver can produce proof of unsatisfiability in the form of a (small) subset of the input clauses that in itself is unsatisfiable. These subsets are commonly referred to as *unsatisfiable cores* [52,47,54]. By using the information provided by the cores, MaxSAT solver can relax soft cores on demand, instead of having to relax all of them upfront. Solvers following this strategy are referred to as *core-guided solvers*. Other proposed methods for MaxSAT solving include incorporating integer linear programming techniques, either as one part of the solving algorithm [55] or by directly encoding the MaxSAT instance as an instance of integer linear programming [56].

In this work, we extend the application domains of MaxSAT to correlation clustering by presenting three different encodings for finding optimal solutions to the correlation clustering problem. Given a symmetric similarity matrix W over a set V of data points (recall Section 2.1), the basic idea behind all of our MaxSAT formulations of correlation clustering is that hard clauses are used to enforce that any solution to the MaxSAT instance represents a well-defined clustering (i.e., a mapping $cl\colon V \to \mathbb{N}$). The soft clauses are used to encode the cost function in a faithful way, so that each solution to the MaxSAT instance can be mapped into a clustering with exactly the same cost. In this way the optimal solution of the created MaxSAT instance can be mapped into the optimal clustering of the correlation clustering problem. Next we will present all three encodings in detail.

6. A MaxSAT formulation of correlation clustering: transitive encoding

Our first MaxSAT formulation, the *transitive encoding*, of correlation clustering can be viewed as a simple reformulation of the integer linear programming formulation (recall Section 3) in terms of MaxSAT.

Similarly as in the ILP formulation, we use boolean variables x_{ij}, where $1 \le i < j \le N$, with the interpretation that $x_{ij} = 1$ iff points v_i and v_j are co-clustered.[2] We again adopt the notational convenience $x_{ij} = x_{ji}$. Now the transitive encoding forms the MaxSAT instance $F^1 = (F_h^1, F_s^1, c)$ summarized in Fig. 5.

We next describe the different parts of F^1 in detail.

[2] Unlike the two other MaxSAT encodings considered in this work, the transitive encoding does not directly allow for enforcing an upper bounds of less than N on the number of clusters.

Hard Clauses F_h^1:	$(\neg x_{ij} \vee \neg x_{jk} \vee x_{ik})$	for all $(v_i, v_j, v_k) \in V^3$ where i, j, k are distinct
Must-Link	(x_{ij})	for all $i < j$ s.t. $W(i, j) = \infty$
Cannot-Link	$(\neg x_{ij})$	for all $i < j$ s.t. $W(i, j) = -\infty$
Soft Clauses F_s^1:	(x_{ij})	for all similar v_i, v_j s.t. $i < j$
	$(\neg x_{ij})$	for all dissimilar v_i, v_j s.t. $i < j$
Cost c of soft clauses	$c((x_{ij})) = W(i, j)$	for all similar v_i, v_j s.t. $i < j$
	$c((\neg x_{ij})) = \lvert W(i, j)\rvert$	for all dissimilar v_i, v_j s.t. $i < j$

Fig. 5. MaxSAT instance $F^1 = (F_h^1, F_s^1, c)$ produced by the transitive encoding.

6.1. Hard clauses

The hard clauses F_h^1 of the transitive encoding are a clausal formulation of the *transitivity constraints* ($x_{ij} + x_{jk} \leq 1 + x_{ik}$ for all distinct i, j, k) of the ILP formulation. In terms of propositional logic, these can be stated as $(x_{ij} \wedge x_{jk}) \rightarrow x_{ik}$, which in clausal form corresponds to

$$\left(\neg x_{ij} \vee \neg x_{jk} \vee x_{ik}\right).$$

6.2. Soft clauses

The soft clauses F_s^1 encode the cost function. Each dissimilar pair of points v_i and v_j ($-\infty < W(i, j) < 0$) that are co-clustered corresponds to exactly one unsatisfied soft clause with weight $-W(i, j)$, and similarly, each similar pair of points v_i and v_j ($\infty > W(i, j) > 0$) that are assigned to different clusters corresponds to one unsatisfied soft clause with weight $W(i, j)$. These conditions are captured by the unit soft clauses $(\neg x_{ij})$ and (x_{ij}), respectively, with weights set to $|W(i, j)|$.

6.3. Encoding constrained clustering

The transitive encoding extends naturally to constrained correlation clustering with ML and CL constraints. For each $W(i, j) = \infty$, v_i and v_j are forced to be co-clustered. This is achieved with the *hard* clause (x_{ij}). Similarly, for each $W(i, j) = -\infty$, points v_i and v_j are forced to different clusters, which is achieved by the hard clause $(\neg x_{ij})$. In addition to ML and CL, various types of other constraints can be expressed.

Example 5 *(Running example of further constraints).* We will use a running example of encoding additional constraints under the three MaxSAT encodings considered in the work, highlighting some of the differences between the encodings. As an example, consider the constraint NotCoClustered(i, j, t) forbidding a triple of points v_i, v_j and v_t from being co-clustered. Under the transitive encoding, this constraint can be encoded as a single clause NotCoClustered$(i, j, t) := (\neg x_{ij} \vee \neg x_{jt} \vee \neg x_{it})$. As another example, consider the cluster-level constraint AtMostInAll(k) requiring each cluster to contain at most k data points. This constraint can be reformulated as requiring that each data point v_i is co-clustered with at most $k-1$ other data points. For a fixed data point v_i the latter formulation can be encoded as a *cardinality constraint* $\sum_{j=\{1,\ldots,N\}\setminus\{i\}} x_{ij} \leq (k-1)$ requiring at most $k-1$ of the variables $x_{i1}, \ldots x_{iN}$ to be set to true, which can further be encoded with hard clauses using one of the several compact cardinality constraints; see e.g. [57,58]. The whole AtMostInAll(k) constraint decomposes in to a conjunction of such cardinality constraints over i.

6.4. Constructing a clustering from a MaxSAT solution to the transitive encoding

Any solution τ to F^1 represents a valid clustering cl_τ of V, constructed in an iterative manner as follows.

While there still are unassigned points left:

1. Let i be the smallest index for which $cl_\tau(v_i)$ is not defined yet and let j be the iteration number ($j = 1\ldots$).
2. Assign $cl_\tau(v_i) = j$.
3. Assign $cl_\tau(v_k) = j$ for all still unassigned v_k for which $\tau(x_{ik}) = 1$.

The fact that cl_τ is well-defined follows from the observation that each point gets assigned to at most one cluster and each iteration of the procedure assigns at least one point to a cluster. Furthermore, the hard transitivity constraints in F^1 ensure that the intended semantics of the x_{ij} variables hold in cl_τ. Hence it follows that the optimal solutions of F^1 correspond to the optimal clusterings of V. The correctness of the transitive encoding can be formalized as follows.

Hard Clauses F_h^2:	$\textsc{ExactlyOne}(i)$	for all $v_i \in V$
	$\textsc{HardSimilar}(i, j, k)$	for all similar v_i, v_j s.t. $i < j$ and $1 \leq k \leq K$
	$\textsc{HardDissimilar}(i, j, k)$	for all dissimilar v_i, v_j s.t. $i < j$ and $1 \leq k \leq K$
Must-Link	$\text{ML}^U(v_i, v_j)$	for all $i < j$ s.t. $W(i, j) = \infty$
Cannot-Link	$\text{CL}^U(v_i, v_j)$	for all $i < j$ s.t. $W(i, j) = -\infty$
Soft Clauses F_s^2:	$\textsc{SoftSimilar}(i, j)$	for all similar v_i, v_j s.t. $i < j$
	$\textsc{SoftDissimilar}(i, j)$	for all dissimilar v_i, v_j s.t. $i < j$
Cost c of soft clauses	$c(\textsc{SoftSimilar}(i, j)) = W(i, j)$	for all similar v_i, v_j s.t. $i < j$
	$c(\textsc{SoftDissimilar}(i, j)) = \lvert W(i, j)\rvert$	for all dissimilar v_i, v_j s.t. $i < j$

Fig. 6. MaxSAT instance $F^2 = (F_h^2, F_s^2, c)$ produced by the unary encoding.

Theorem 2. *Given a set V of data points and a symmetric similarity matrix W over V, let F^1 be the MaxSAT instance produced by the transitive encoding on W. The clustering $cl_{\tau^*}\colon V \to \mathbb{N}$ constructed from an optimal solution τ^* to F^1 is an optimal clustering of V.*

A detailed proof of the theorem is given in Appendix A.

We note that the transitive encoding does not require a predefined number of clusters. This is avoided by the definition of the x_{ij} variables, interpreted as pairwise indicator variables for two data points v_i, v_j being assigned to the same cluster. However, the encoding is not very compact. Its size is similar to the ILP presented earlier, $\mathcal{O}(N^2)$ variables and $\mathcal{O}(N^3)$ clauses, suggesting that also this encoding does not scale well. Next we will present a *unary encoding* of correlation clustering into MaxSAT, which to some extent addresses the compactness issue of the transitive encoding.

7. An unary encoding of correlation clustering into MaxSAT

We now consider a more compact *unary encoding*, which to some extent resembles the quadratic integer programming formulation presented in Section 4. Similarly to the QIP, the unary encoding allows an upper bound K on the number of available clusters. By letting $K = N$, the set of clusterings produced by the unary encoding is exactly the same as for the transitive encoding. The size of the unary encoding is $\mathcal{O}(E \cdot K + N \cdot K)$ variables and $\mathcal{O}(E \cdot K)$ clauses where E is the number of non-zero values in the input similarity matrix W. Due to the dependence on E, in practice the unary encoding is more compact than the transitive encoding whenever the input matrix contains 0-entries or $K < N$.

The unary encoding involves $N \cdot K$ boolean variables y_i^k, where $i = 1..N$ (the number of data points) and $k = 1..K$ (the number of clusters). The intended interpretation of these variables is that $y_i^k = 1$ iff point v_i belongs to cluster k. Furthermore, the encoding employs two types of auxiliary variables.

- A_{ij}^k, where $i = 1..N$, $j = 2..N$, $i < j$, $W(i, j) > 0$, and $k = 1..K$, with the interpretation $A_{ij}^k = 1$ iff points v_i and v_j are both assigned to cluster k.
- D_{ij}, where $i = 1..N$, $j = 2..N$, $i < j$, and $W(i, j) < 0$, with the interpretation that if $D_{ij} = 0$, then points v_i and v_j are assigned to different clusters.

These variables are used for compactly encoding the similarity and dissimilarity constraints. We will next present details on the clauses used in the unary encoding. As with the transitive encoding, the hard clauses limit the set of solutions to well-defined clusterings, and the soft clauses encode the cost function in a faithful way. However, the hard and soft clauses differ significantly from the clauses in the transitive encoding. Most notably, both hard and soft clauses are included in the unary encoding for encoding the similarity and dissimilarity constraints.

Concretely, the unary encoding forms the MaxSAT instance $F^2 = (F_h^2, F_s^2, c)$ summarized in Fig. 6.

We next describe the different parts of F^2 in detail.

7.1. Ensuring well-defined clusterings

The hard constraints $\textsc{ExactlyOne}(i)$ constrain the search to well-defined clusterings by enforcing that each data point v_i is assigned into exactly one cluster k. In terms of the variables in the encoding this means that, for each i, exactly one of the variables $y_i^1, \ldots, y_i^K$ should be assigned to 1, i.e.,

$$\textsc{ExactlyOne}(i) := \sum_{k=1}^{K} y_i^k = 1.$$

A number of different encodings of this cardinality constraint as clauses have been previously developed [59]. In our experiments, we used the so-called *sequential encoding* [60] which is linear, or more precisely, introduces $3K - 4$ clauses and $K - 1$ auxiliary variables for each i. We refer the interested reader to [60] for a detailed description of this encoding.

7.2. Encoding similarity

For a similar pair of data points v_i and v_j, the constraints $\textsc{HardSimilar}(i, j, k)$ for each $k = 1..K$ and $\textsc{SoftSimilar}(i, j)$ together enforce the requirement that v_i and v_j are assigned to the same cluster whenever the soft constraint $\textsc{SoftSimilar}(i, j)$ is satisfied. In terms of propositional logic, this requirement can be expressed as the formula

$$(y_i^1 \wedge y_j^1) \vee (y_i^2 \wedge y_j^2) \vee \ldots \vee (y_i^K \wedge y_j^K).$$

In order to the express this propositional formula as clauses, we employ the auxiliary variables A_{ij}^k and define the semantics of these to be $\tau(A_{ij}^k) = 1$ iff $\tau(y_i^k \wedge y_j^k) = 1$. In terms of propositional logic, the defining constraint is $A_{ij}^k \leftrightarrow (y_i^k \wedge y_j^k)$, which can be expressed as

$$\textsc{HardSimilar}(i, j, k) := \{(\neg A_{ij}^k \vee y_i^k), (\neg A_{ij}^k \vee y_j^k), (A_{ij}^k \vee \neg y_i^k \vee \neg y_j^k)\}.$$

We note that the definitions of the auxiliary variables do not yet enforce points v_i and v_j to be assigned to cluster k. Instead, the clauses $\textsc{HardSimilar}(i, j, k)$ state that the variable A_{ij}^k is set to true if and only if points v_i and v_j are both assigned to cluster k. This must hold in every solution to F^2, hence the clauses are *hard*.

Using the auxiliary variables, the soft constraint expressing that the points v_i and v_j are assigned to the same cluster can be encoded as the clause

$$\textsc{SoftSimilar}(i, j) := (A_{ij}^1 \vee \cdots \vee A_{ij}^K) \text{ with weight } c(\textsc{SoftSimilar}(i, j)) = W(i, j).$$

For some intuition, we note that if this clause is satisfied in a solution τ, then for some k, $\tau(A_{ij}^k) = 1$. Since all hard clauses are satisfied in any solution, it follows that points v_i and v_j will be assigned to cluster k, exactly as required. Similarly, if points v_i and v_j are not assigned to the same cluster, then due to the hard constraints we have $\tau(A_{ij}^k) = 0$ for all k, and the soft clause will not be satisfied. Each unsatisfied clause must increase the cost of a MaxSAT solution according to the similarity values of the corresponding points, which is why the weight of the clause is set to $W(i, j)$.

7.3. Encoding dissimilarity

For a dissimilar pair of data points v_i and v_j, the clauses $\textsc{HardDissimilar}(i, j, k)$ for each $k=1..K$ and $\textsc{SoftDissimilar}(i, j)$ together enforce the requirement that v_i and v_j are assigned to different clusters. This can be expressed by requiring for each cluster that at least one of v_i and v_j should not be assigned to that cluster, which in clausal form is expressed by $(\neg y_i^k \vee \neg y_j^k)$ for a cluster k. The whole constraint enforcing v_i and v_j to be assigned to different clusters is hence

$$(\neg y_i^1 \vee \neg y_j^1) \wedge \ldots \wedge (\neg y_i^K \vee \neg y_j^K). \tag{5}$$

Equation (5) is already in clausal form. However, we want to make sure that breaking any of the individual clauses corresponds to a cost of $|W(i, j)|$. To achieve this, we use the auxiliary variables D_{ij}, and define them in terms of propositional logic as $\neg D_{ij} \rightarrow (\neg y_i^k \vee \neg y_j^k)$ for each cluster $k = 1..K$. That is, if $\tau(D_{ij}) = 0$ for some solution τ to F^2, then v_i and v_j are not assigned to the same cluster.[3] The defining constraint can be expressed as the hard clauses

$$\textsc{HardDissimilar}(i, j, k) := (D_{ij} \vee \neg y_i^k \vee \neg y_j^k).$$

The auxiliary variable D_{ij} makes it possible to express the soft constraint requiring v_i and v_j to not be co-clustered simply as

$$\textsc{SoftDissimilar}(i, j) := (\neg D_{ij}) \text{ with weight } c(\textsc{SoftDissimilar}(i, j)) = |W(i, j)|.$$

For some intuition, we have that if the clause $(\neg D_{ij})$ is satisfied in a solution to F^2, then the clauses $(\neg y_i^k \vee \neg y_j^k)$ also have to be satisfied for all k. Hence points v_i and v_j are not assigned to the same cluster. On the other hand, if v_i and v_j are assigned to the same cluster k, then the solution has to assign $D_{ij} = 1$ in order to satisfy the hard clause $(D_{ij} \vee \neg y_i^k \vee \neg y_j^k)$, resulting in one unsatisfied clause with weight $|W(i, j)|$, exactly as required for representing the correlation clustering cost function faithfully.

[3] The formalism behind grouped soft clauses like the ones in Equation (5) is known as *group-MaxSAT*. An exact treatment of group-MaxSAT is beyond the scope of this work, we refer the interested reader to [61].

7.4. Encoding constrained clustering

By noticing that for each $k = 1..K$ we need to enforce that $cl(v_i) = k$ iff $cl(v_j) = k$, the must-link constraint over v_i and V_j can be encoded under the unary encoding as

$$\mathrm{ML}^U(v_i, v_j) := \{(\neg y_i^1 \vee y_j^1), (y_i^1 \vee \neg y_j^1), \ldots, (\neg y_i^K \vee y_j^K), (y_i^K \vee \neg y_j^K)\},$$

where the clauses $(\neg y_i^k \vee y_j^k)$ and $(y_i^k \vee \neg y_j^k)$ correspond to $y_i^k \leftrightarrow y_j^k$. For some intuition, in any solution τ we have that, whenever $\tau(y_i^k) = 1$, the solution has to assign $\tau(y_j^k) = 1$ in order to satisfy $(\neg y_i^k \vee y_j^k)$. Furthermore, based on the other hard clauses, we know that there exists exactly one $k = 1..K$ for which $\tau(y_i^k) = 1$, and hence $\tau(y_i^{k'}) = 0$ for all $k' \neq k$. Thus τ has to assign $\tau(y_j^{k'}) = 0$ in order to satisfy the clause $(y_i^{k'} \vee \neg y_j^{k'})$; hence the points are assigned to the same cluster.

The benefit of encoding the must-link constraint in this way compared to the similarity constraint presented earlier is the elimination of the auxiliary variables A_{ij}^k and hence a decrease in the number of clauses generated. On the other hand, similarity constraints cannot be encoded directly in this way since they are soft. Furthermore, whenever a similarity constraint is not satisfied, the cost added to a MaxSAT solution should be exactly the corresponding similarity value, which is controlled in a simple way with the A_{ij}^k variables.

Cannot-link constraints in the unary encoding can also be encoded more compactly than the dissimilarity constraints. The variable D_{ij} in the encoding is used to ensure that an unsatisfied dissimilarity constraint corresponds exactly to cost $|W(i,j)|$. If we know that the dissimilarity constraint has to be satisfied (making it a hard cannot-link constraint), we can simply leave out the extra variable. The intuition between the cannot-link clauses is that for each $k = 1..K$ and any solution τ, either $\tau(y_i^k) = 0$ or $\tau(y_j^k) = 0$. Stated as clauses, we have

$$\mathrm{CL}^U(v_i, v_j) := \{(\neg y_i^1 \vee \neg y_j^1), \ldots, (\neg y_i^K \vee \neg y_j^K)\}.$$

Example 6 *(Running example of further constraints continued).* Under the unary encoding, the NotCoClustered(i, j, t) constraint, forbidding all three of the points v_i, v_j and v_t from being co-clustered, can be encoded with a set of K clauses

$$\textsc{NotCoClustered}(i, j, t) := \{(\neg y_i^1 \vee \neg y_j^1 \vee \neg y_t^1), \ldots, (\neg y_i^K \vee \neg y_j^K \vee \neg y_t^K)\}.$$

The constraint includes one clause for each cluster $s = 1..K$, each forbidding all three points from being assigned to cluster s. The constraint AtMostInAll(k), requiring each cluster to contain at most k data points, can be encoded as a conjunction of K cardinality constraints, namely, by enforcing $\sum_{i=1}^{N} y_i^j \leq k$ over each cluster index j.

7.5. Constructing a clustering from a MaxSAT solution to the unary encoding

Given a solution τ to F^2, we can easily construct a corresponding well-defined clustering cl_τ of the data points by assigning each point v_i into the cluster k for which $\tau(y_i^k) = 1$. Due to the hard constraints F_h^2, in any solution τ there is exactly one such k for every i. Especially, the clustering constructed from an optimal solution to F^2 will be an optimal clustering of the data, minimizing the correlation clustering objective function. This correctness of the unary encoding can be formalized as follows.

Theorem 3. *Given a set V of data points with $|V| = N$, a symmetric similarity matrix W over V, and an upper limit K on the available clusters such that $1 \leq K \leq N$, let F^2 be the MaxSAT instance produced by the unary encoding on W. The clustering $cl_{\tau^*}: V \to \{1, \ldots K\}$ constructed from an optimal solution τ^* to F^2 is an optimal clustering of V over all clusterings $cl: V \to \{1, \ldots K\}$. In other words, cl_{τ^*} is optimal over all clusterings of V that use at most K clusters.*

Intuitively, the theorem follows from the already discussed connections between cost incurred by a clustering and the weight of unsatisfied soft clauses in the unary encoding. A proof of Theorem 3 is provided in Appendix A.

8. A binary encoding of correlation clustering into MaxSAT

As the third encoding, we describe a *binary encoding* of correlation clustering as MaxSAT, which is essentially a bitwise reformulation of the unary encoding. Similarly to the unary encoding, the binary encoding allows an upper limit K on the available clusters. As is often the case with SAT and MaxSAT encodings, the binary encoding is more compact than both the unary and the transitive encoding, regardless of the input similarity matrix or the value of K. An instance formed by the binary encoding contains $\mathcal{O}(E + N \cdot \log_2 K)$ variables and $\mathcal{O}(E \cdot \log_2 K)$ clauses, where E is the number of non-zero values in the input similarity matrix W.

For simplicity, we first assume that K is a power of 2, more precisely $K = 2^a$ for some $a \in \mathbb{N}$. From this it follows that $\log_2 K = a$ is an integer. The encoding also works if this is not the case; we will describe the required adaptations in

Hard Clauses F_h^3:	$\textsc{Equality}(i, j, k)$	for all $W(i, j) \in \mathbb{R} \setminus \{0\}$
	$\textsc{SameCluster}(i, j)$	for all $W(i, j) \in \mathbb{R} \setminus \{0\}$
Must-Link	$\mathrm{ML}^B(v_i, v_j)$	for all $i < j$ s.t. $W(i, j) = \infty$
Cannot-Link	$\mathrm{CL}^B(v_i, v_j)$	for all $i < j$ s.t. $W(i, j) = -\infty$
If $K \neq N$ and $K \neq 2^a$ for any a	$\textsc{ClustersLessThan}(i, K)$	for all $v_i \in V$
Soft Clauses F_s^3:	(S_{ij})	for all similar v_i, v_j s.t. $i < j$
	$(\neg S_{ij})$	for all dissimilar v_i, v_j s.t. $i < j$
Cost c of soft clauses	$c((S_{ij})) = W(i, j)$	for all similar v_i, v_j s.t. $i < j$
	$c((\neg S_{ij})) = \lvert W(i, j)\rvert$	for all dissimilar v_i, v_j s.t. $i < j$

Fig. 7. MaxSAT instance $F^3 = (F_h^3, F_s^3, c)$ produced by the binary encoding.

Section 8.5. The encoding uses a variables b_i^k where $1 \leq k \leq a$ for each point v_i. The intended semantics of these variables is that point v_i is assigned to cluster index $b_i^a..b_i^1$, interpreted as a binary number with the least significant bit to the right. Additionally, we employ two types of auxiliary variables.

- EQ_{ij}^k, where $1 \leq i < j \leq N$, $W(i, j) \in \mathbb{R} \setminus \{0\}$, and $1 \leq k \leq a$. The intended semantics of EQ_{ij}^k is $EQ_{ij}^k = 1$ iff $b_i^k = b_j^k$.
- S_{ij}, where $1 \leq i < j \leq N$ and $W(i, j) \in \mathbb{R} \setminus \{0\}$. $S_{ij} = 1$ iff points v_i and v_j are co-clustered. (Note the equivalence: also, if $S_{ij} = 0$, then points v_i and v_j are not assigned to the same cluster.)

An instance F^3 produced by the binary encoding is summarized in Fig. 7. This time, the only hard clauses required are the clauses defining the auxiliary variables. This is due to the fact that any MaxSAT solution has to assign all the variables b_i^k in some unique way, and hence any solution will represent a well-defined clustering. We next describe the binary encoding in more detail.

8.1. Hard clauses

As the b_i^k variables form the bit-representation of the cluster index of point v_i, the question of whether two points v_i and v_j are assigned to the same cluster is equivalent to whether the values of b_i^k and b_j^k are equal for all $1 \leq k \leq a$. In order to reason about the equality of individual bits, the binary encoding uses a "equality" variables EQ_{ij}^k for each pair of points v_i and v_j for which $i < j$ and $W(i, j) \in \mathbb{R} \setminus \{0\}$. These variables are defined to be equivalent to $\tau(b_i^k) = \tau(b_j^k)$ when τ is a solution to F^3. In terms of propositional logic, the defining constraint is $EQ_{ij}^k \leftrightarrow (b_i^k \leftrightarrow b_j^k)$, which corresponds to the set of clauses

$$\textsc{Equality}(i, j, k) := \{(EQ_{ij}^k \vee b_i^k \vee b_j^k), (EQ_{ij}^k \vee \neg b_i^k \vee \neg b_j^k), (\neg EQ_{ij} \vee \neg b_i^k \vee b_j^k), (\neg EQ_{ij} \vee b_i^k \vee \neg b_j^k)\}.$$

Encoding the semantics of the S_{ij} variables is straightforward using the equality variables. Two points v_i and v_j are assigned to the same cluster iff the values at each bit-position in the bit representation of their cluster indices are the same. Stated in propositional logic, we have $S_{ij} \leftrightarrow (EQ_{ij}^1 \wedge \ldots \wedge EQ_{ij}^a)$, which corresponds to

$$\textsc{SameCluster}(i, j) := \{(\neg S_{ij} \vee EQ_{ij}^1), \ldots, (\neg S_{ij} \vee EQ_{ij}^a), (S_{ij} \vee \neg EQ_{ij}^1 \vee \ldots \vee \neg EQ_{ij}^a)\}.$$

8.2. Soft clauses

As the variable S_{ij} has the exact same semantics as the variable x_{ij} in the transitive encoding, it can be used to formulate the soft clauses of the binary encoding in a very similar manner as the soft clauses in the transitive encoding. For every similar pair of points v_i and v_j, the cost of the clustering should increase by $W(i, j)$ whenever the points are not assigned to the same cluster. This condition is encoded by the unit soft clause (S_{ij}) with weight $c((S_{ij})) = W(i, j)$. Analogously, for every dissimilar pair the instance includes the soft clause $(\neg S_{ij})$ with weight $c((\neg S_{ij})) = |W(i, j)|$.

8.3. Encoding constrained clustering

For compactly encoding the must-link constraint in the binary encoding, we simplify the similarity constraint. We need to ensure that $\tau(b_i^k) = \tau(b_j^k)$ for all bits $k = 1..a$ and all MaxSAT solutions τ. For a fixed k, this can be stated as $(b_i^k \leftrightarrow b_j^k)$, which as clauses is expressed by $(\neg b_i^k \vee b_j^k), (b_i^k \vee \neg b_j^k)$. Hence the whole must-link constraint is

$$\mathrm{ML}^B(v_i, v_j) := \{(\neg b_i^1 \vee b_j^1), (b_i^1 \vee \neg b_j^1), \ldots, (\neg b_i^a \vee b_j^a), (b_i^a \vee \neg b_j^a)\}.$$

The cannot-link constraint can be seen as a simplified dissimilarity constraint. The variable EQ^k_{ij} and the clauses defining it are still required for all bits. However, the cannot-link constraint can be stated as a single clause: we simply require that there exists a bit-position k such that the values b^k_i and b^k_j differ. The whole cannot-link constraint is

$$\mathrm{CL}^B(v_i, v_j) := \{\textsc{Equality}(i, j, 1), \ldots, \textsc{Equality}(i, j, a), (\neg EQ^1_{ij} \vee \ldots \vee \neg EQ^a_{ij})\}.$$

Example 7 *(Running example of further constraints continued).* Due to the similar semantics of the S_{ij} variables of the binary encoding and the x_{ij} variables of the transitive encoding, both of our example constraints can be encoded very similarly to the transitive encoding. The $\textsc{NotCoClustered}(i, j, t)$ constraint, forbidding all three of the points v_i, v_j and v_t from being co-clustered, can be encoded by a single clause $\textsc{NotCoClustered}(i, j, t) := (\neg S_{ij} \vee \neg S_{it} \vee \neg S_{jt})$, i.e., more compactly than in the unary encoding directly. Also, the $\textsc{AtMostInAll}(k)$ constraint can be encoded similarly to the transitive encoding by using for each v_i, a cardinality constraint forbidding v_i from being co-clustered with mode than $k-1$ other points: $\sum_{j=\{1\ldots N\}\setminus\{i\}} S_{ij} \le k-1$.

8.4. Constructing a clustering from a MaxSAT solution to the binary encoding

Given a solution τ to F^3, there is again a very natural way of constructing a clustering of V. For each data point v_i, let $\tau(b^a_i)\tau(b^{a-1}_i)\ldots\tau(b^1_i) = c$, where the left hand side is interpreted as a binary number, and assign $cl_\tau(v_i) = c+1$. Since the number of available bits is $\log_2 K$, it follows that $0 \le c \le K-1$, and hence $1 \le cl_\tau(v_i) \le K$ holds for all v_i. The clustering constructed from an optimal solution to F^3 is optimal amongst all clusterings using at most K clusters.

Theorem 4. *Given a set V of data points with $|V| = N$, a symmetric similarity matrix W over V, and an upper limit K on the available clusters such that $1 \le K \le N$, let F^3 be the MaxSAT instance produced by the binary encoding on W. The clustering $cl_{\tau^*}\colon V \to \{1, \ldots K\}$ constructed from an optimal solution τ^* to F^3 is an optimal clustering of V under W over all clusterings $cl\colon V \to \{1, \ldots K\}$. In other words, cl_{τ^*} is optimal over all clusterings of W that use at most K clusters.*

A proof of this theorem is provided in Appendix A.

8.5. The binary encoding for general K

So far we have assumed that the upper limit on the available clusters is a power of 2, or, more precisely, that $K = 2^a$ for some a. This assumption simplifies the binary encoding since the values representable in binary with a bits are exactly 0 to $2^a - 1$. It is also possible to constraint K to an arbitrary value. A simple approach would be to encode a separate constraint for each point v_i and each value $j \in \{K, K+1, \ldots, 2^a - 1\}$ forbidding the value of the bit variables $b^a_i, \ldots, b^1_i$ (interpreted as a binary number) from being equal to j. However, this would result in $\mathcal{O}(N^2 \cdot \log_2 N)$ clauses, the same as the worst-case size of the whole encoding.

A more compact formulation can be obtained by observing that, for each data point we only need to encode a single constraint stating that the value of its assigned cluster index should be *less than* K. For a given K, let K^j denote the value of the jth bit in the binary representation of K. Note that as $2^{a-1} < K \le 2^a$, there are exactly a bits in the binary representation of K. For any set of bit variables $b^a_i, \ldots, b^1_i$, denote the value represented by these variables in binary by $(b^a_i \ldots b^1_i)_2$. For a given datapoint v_i we can encode the constraint $(b^a_i \ldots b^1_i)_2 < K$ recursively using the observation that a binary number $(b^a_i \ldots b^1_i)_2$ is less than another binary number $(K^a \ldots K^1)_2$ iff

- $K^a = 1$ and $b^a_i = 0$, or
- $K^a = b^a_i$ and $(b^{a-1}_i \ldots b^1_i)_2 < (K^{a-1} \ldots K^1)_2$.

This formulation of inequality between binary numbers follows directly from the properties of binary numbers. We encode it as MaxSAT by introducing a fresh variables B^j_i, $1 \le j \le a$, and adding clauses defining them recursively as

$$\begin{aligned}
\textsc{DefB}(i, 1) &:= B^1_i \leftrightarrow \left(\neg b^1_i \wedge (K^1 = 1)\right), \\
\textsc{DefB}(i, j) &:= B^j_i \leftrightarrow \left((\neg b^j_i \wedge (K^j = 1)) \vee ((b^j_i \leftrightarrow K^j) \wedge B^{j-1}_i)\right).
\end{aligned} \tag{6}$$

As the value of K is known, we can simplify the definition accordingly when adding the clauses to the encoding. Using these variables, the whole constraint limiting the number of clusters is enforced by the clauses defining the semantics of the B^j_i variables, together with N unit clauses, one for each data point:

$$\textsc{ClustersLessThan}(i, K) := \{\textsc{DefB}(i, 1), \ldots, \textsc{DefB}(i, a) \mid i = 1..N\} \cup \{(B^a_1), \ldots, (B^a_N)\}.$$

The size of this formulation is $\mathcal{O}(N \cdot \log_2(N))$.

Table 1
Number of data points in datasets considered.

Dataset	Number of points
Ecoli	327
Ionosphere	351
ORL	400
Prot3	567
Umist	575
Prot2	586
Prot4	654
Prot1	669
Breastcancer	683
Diabetes	768
Vowel	990

9. Experimental evaluation

We will now describe an experimental evaluation of our MaxSAT-based approach to correlation clustering.

9.1. Benchmarks

We experiment on real-world datasets consisting of similarity values between amino-acid sequences of different proteins [62], as well as similarity matrices we obtained from standard UCI benchmark datasets. For each of the obtained similarity matrices, we normalized the matrix entries to the range $[-0.5, 0.5]$.

9.1.1. Protein sequence datasets

We obtained four protein sequence datasets from http://www.paccanarolab.org/scps. The data consists of similarity values between amino-acid sequences, originally computed using BLAST [63]. All values were originally in the range $[0, 1.0]$. Normalization of the similarity information to the range $[-0.5, 0.5]$ was done by subtracting 0.5 from each entry. Table 1 shows the number of data points for each data set.

9.1.2. UCI datasets

In addition to the protein sequence datasets, we produced similarity matrices based on the following UCI datasets.

- ORL: the AT&T ORL database of images of faces, each of size 92×112. Obtained from http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.
- Ionosphere: the UCI ionosphere dataset, for classification of radar returns from the ionosphere, originally with 34 attributes. Obtained from http://archive.ics.uci.edu/ml/.
- Umist: the Sheffield (previously UMIST) Face Database, each face image of size 92×112. Obtained from http://www.sheffield.ac.uk/eee/research/iel/research/face.
- Breastcancer: the LIBSVM breast-cancer dataset, originally named "Wisconsin Breast Cancer in UCI". The set contains 10 features. Obtained from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.
- Diabetes: the LIBSVM diabetes dataset, originally from UCI, containing 8 features. Obtained from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.
- Ecoli: the UCI Ecoli dataset, containing protein localization sites, with 8 features. Obtained from http://archive.ics.uci.edu/ml/.
- Vowel: the LIBSVM Vowel dataset, originally from UCI, with 10 features. Obtained from http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

For these datasets, we first calculated the normalized Euclidean distance between each pair of points, and directly interpreted the distances as similarity values by linear inverse mapping to the range $[-0.5, 0.5]$. In order to simulate incomplete similarity information, we finally modified all similarity values in the range $[-0.25, 0.25]$ to be 0. The size of each dataset is reported in Table 1.

9.1.3. Setup

For solving the MaxSAT instances resulting from our encodings, we used the academic off-the-shelf MaxSAT solver MaxHS [64,55,65] (MaxSAT evaluation 2013 version) obtained from the authors. MaxHS implements a hybrid approach to MaxSAT solving, combining the logical reasoning power of a SAT solver with the arithmetic reasoning power of an integer linear programming solver. During its execution, MaxHS maintains a set of unsatisfiable cores (recall Section 5.2). At each iteration, the ILP solver is used for finding a *minimum-cost hitting set* over the soft clauses in the current set of cores. Clauses in the hitting set are then temporarily removed from the instance and the SAT solver is invoked again. MaxHS terminates when the working formula is satisfiable, at which point the assignment returned by the SAT solver is an optimal solution to the MaxSAT instance. We note that MaxHS is by no means the only possible choice for a MaxSAT solver to use. We also report on a comparison of different state-of-the-art MaxSAT solvers in Section 9.4.3, the results of which motivate the use of MaxHS.

We compare the MaxSAT-based approach with exactly solving the integer linear programming and the quadratic integer programming formulations of correlation clustering (recall Sections 3 and 4, respectively). We used the commercial state-of-the-art integer programming solvers IBM CPLEX (version 12.6) and Gurobi Optimizer (version 6.0) for solving the integer linear programs, and additionally, the non-commercial SCIP [66] framework for solving the quadratic integer programs. Furthermore, we also compare to two approximative algorithms in terms of the cost of solutions obtained: the approximation algorithm KwickCluster (KC) proposed in [5] and further considered in [67], and the SDPC approach based on a semi-definite relaxation of the quadratic integer programming formulation, proposed in [34]. More details on these algorithms are provided in Section 10.1. For solving the semi-definite programs, we used the Matlab package SeDuMi 1.3 [68].

On the protein data we also experimented with the algorithms described in [62], available from http://www.paccanarolab.org/scps, which are specialized algorithms for correlation clustering protein sequences. The authors provide two algorithms that allow an unrestricted number of clusters by default. One is based on spectral clustering (SCPS) and the other on connected component analysis (CCA).

In addition to the comparative results, we also report on MaxSAT-specific experiments on the effect of MaxSAT-level preprocessing (in Section 9.4.1) and symmetry breaking (in Section 9.4.2) on solving times. We employed MaxSAT preprocessing in all experiments due to its positive impact on solving times. As for symmetry breaking on the MaxSAT-level in the other experiments, we only applied partial symmetry breaking to all formulas by enforcing the point with the lowest index to always be assigned to the first cluster.

A timeout of 8 hours and a memory limit of 30 GB were enforced on each individual run of a solver. The experiments were run under Linux on eight-core Intel Xeon E5440 2.8-GHz cluster nodes each with 32 GB of RAM. In order to ensure repeatable results, only a single algorithm on a single benchmark instance was executed on each cluster node at each time.

9.2. *Experiments on unconstrained correlation clustering*

We first focus on unconstrained correlation clustering, i.e., correlation clustering under the assumption that there are no infinite values in the input similarity matrices.

9.2.1. *Comparison of algorithms providing optimal solutions*

We start with a comparison of the exact approaches to correlation clustering: our three MaxSAT encodings, the integer linear programming formulation (ILP), and the quadratic integer programming formulation (QIP). As the size of the transitive encoding and the integer linear program does not depend on the number of non-zero elements in the similarity matrix, for these experiments we created instances by varying the number of points $n \geq 50$ in the four protein datasets (Prot1, Prot2, Prot3 and Prot4) by considering only the n first rows and columns of the original similarity matrix of each data set.

The results are shown in Fig. 8. The reason for the absence of the QIP approach from the plot is that neither CPLEX, Gurobi, nor SCIP was able to solve any of the quadratic programs exactly within the time limit. For example, SCIP was able to solve the instances when using 20 points within seconds, but was unable to solve 50 points within 8 hours. While we do not have a definitive explanation for this poor behavior, one possible explanation may deal with the non-convexity (recall Section 4) of the QIP formulation of correlation clustering.[4] The transitive and the unary MaxSAT encodings, as well as the ILP approach, are competitive with the binary encoding only when the number of points is small. However, all three MaxSAT encodings scale better than ILP and QIP. Both CPLEX and Gurobi ran out of memory on the ILP formulation for instances larger than 300 points, suggesting it will fail to solve larger instance irrespective of the timeout. Furthermore, the encodings for which the size of the instance is not dependent on the number of non-zero entries in the similarity matrix cannot benefit from any sort of pruning that one might be able to do on the similarity values of the input data.

Based on these observations, for the MaxSAT-based approach we focus on the binary encoding in the rest of the experiments.

9.2.2. *Performance on sparse data*

Next we simulate a setting in which the input data is sparse, that is, situations in which the similarity information available is incomplete. For $p \in \{0.05, 0.10, \ldots, 1\}$, we created instances from a given similarity matrix W by independently setting each non-zero element $W(i, j)$ to 0 with a probability $1 - p$. This results in a matrix W' where the expected number of non-zero entries is $p \cdot 100\%$ of the number of non-zero entries in W.

We ran each of the approximative algorithms 100 times on each instance, and report the best values returned by them. We note that a single run of any of the approximative algorithms is very short, at most one minute for SDPC and within 10 seconds for the others.

Figs. 9, 10 and 11 summarize the result of solving the sparse instances. A sparser matrix results in MaxSAT instances which are faster to solve. More importantly, however, we notice that MaxSAT is fairly robust when it comes to dealing with

[4] This would be inline with behavior observed in other problem domains as well, see e.g. [69].

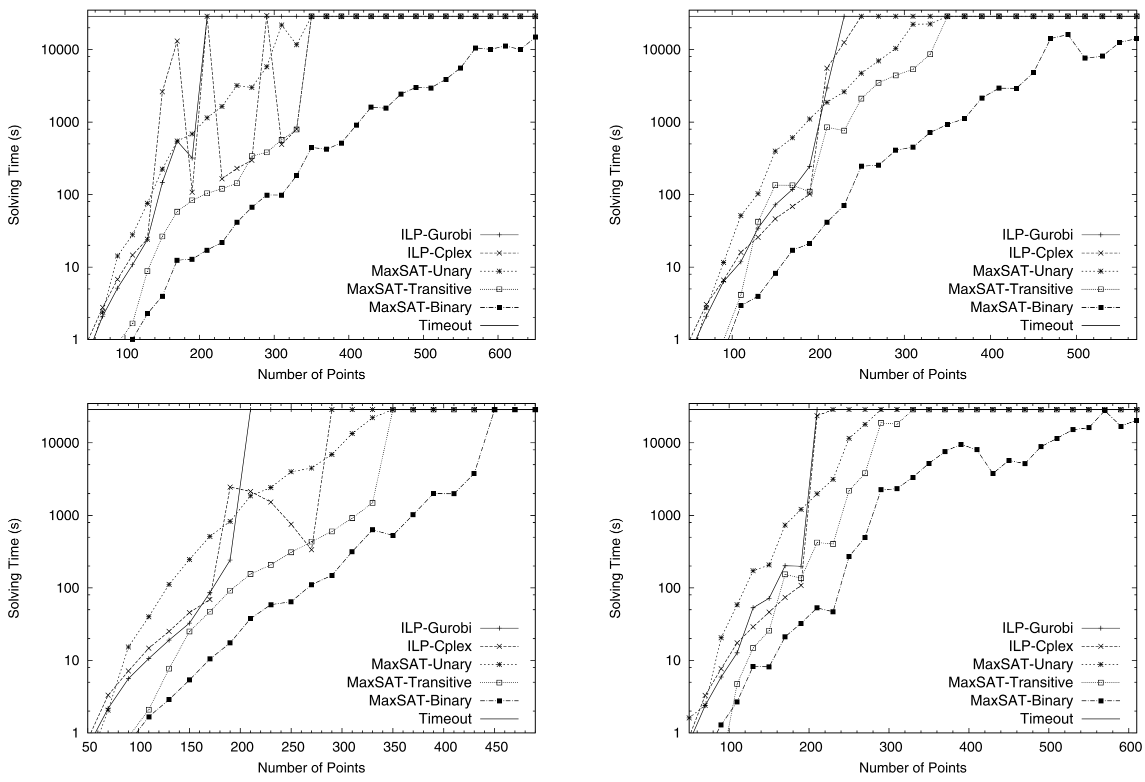

Fig. 8. Point scalability of the exact approaches. Top left: Prot1, top right: Prot2, bottom left: Prot3, bottom right: Prot4.

sparse data and the cost of the solution clustering. We experimentally compare the robustness of the different algorithms by calculating the cost $H(W, cl)$ for clusterings cl which were obtained with W' as input. This simulates a setting where there is some "true" objective function value that we would like the algorithms to optimize, but the amount of information available to the algorithms is limited/noisy. The cost of clusterings produced by the binary encoding is significantly lower than the other generic correlation clustering algorithms KwickCluster and SDPC for all values of p solvable by MaxSAT. For $p > 0.4$, the solutions obtained with MaxSAT have a clearly lower cost than the solutions provided by two algorithms specialized for clustering protein sequences. Perhaps the most significant observation here is that, whenever the dataset was not solvable within the timeout, the clusterings obtained by MaxSAT on the highest value of p still solvable had in general a lower cost than any of the approximative algorithms at $p = 1.0$. This suggests that one can prune away a significant number of the non-zero entries in a matrix, hence speeding up MaxSAT solving, and still obtain clusterings of lower cost than those obtained with the approximative algorithms. We hypothesize that a more sophisticated method of pruning, perhaps taking into account the structure of the input matrix, could further improve the results. Comparing KwickCluster with SDPC, we observe that semi-definite programming performs slightly better on extremely sparse instances. However, when the density of the underlying graph increases, the performance of KwickCluster improves while the performance of SDPC remains fairly constant. One possible explanation for this could be that the relaxation of a quadratic program into a semi-definite program (see Section 10.1 for details) has a similar effect to the quality of the obtained clustering as pruning similarity information from the matrix.

9.3. Constrained correlation clustering

We now turn our attention to MaxSAT-based constrained correlation clustering.

9.3.1. Instance-level constraints

We first consider a situation in which an oracle, for example a domain expert, provides domain specific knowledge in the form of a set of must-link and cannot-link constraints the solution clusterings are expected to satisfy. By running several tests with an increasing number of constraints, we simulate a setting in which the current solution clustering is shown to the oracle, who is then allowed to add more constraints to the clustering algorithm in order to further restrict the set of

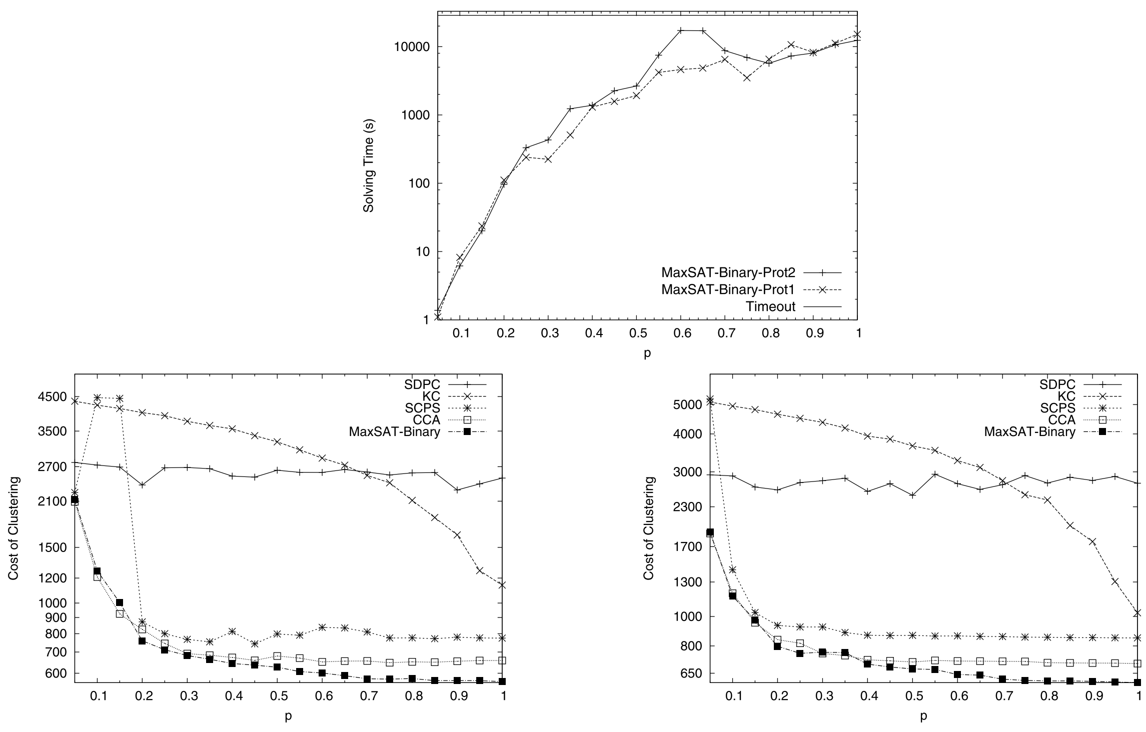

Fig. 9. Top: Evolution of running times. Bottom: Cost of the clusterings obtained on sparse matrices. Bottom left: Prot1, bottom right: Prot2.

acceptable clusterings. An iterative setting like this has previously been studied for example in [70,71] and has been shown to greatly increase the clustering accuracy in other clustering problems [70,71,27].

We simulate this setting with the help of a (human created) "golden" clustering supplied with each of the datasets. Given the similarity matrix W based on a dataset, the golden clustering can be seen as a symmetric similarity matrix G_W of the same dimension where each element is either ∞ or $-\infty$. To simulate this iterative setting, we sampled an increasing number of pairs of indices $i < j$, and modify W by assigning $W(i, j) = G_W(i, j)$. Added x% user knowledge (UK) means that x% of available pairs of indices $i < j$ were sampled. This results in a setting in which at each iteration the MaxSAT algorithm has an increasing amount of information on the golden clustering. We note that, to the best of our knowledge, the considered approximative algorithms cannot handle such a constrained correlation clustering setting directly. Even though additional constraints could be included into the semi-definite program solved with SDPC, there are no guarantees that the clustering obtained after the rounding procedure within SDPC respects the added constraints (see Section 10.1 for more details). This is why all the values reported for those are for 0% added UK.

In these tests the performance of our encoding is evaluated using the well-known *rand index* [72] designed for measuring the similarity of two clusterings.

Definition 1. Given a dataset $V = \{v_1 \dots v_N\}$, a clustering cl of V, and an example clustering g, let

$$TP = \left|\{(v_i, v_j) \mid cl(v_i) = cl(v_j) \wedge g(v_i) = g(v_j)\}\right|$$

denote the number of pairs of points $i < j$ that are co-clustered in both cl and g (true positives). Let

$$TN = \left|\{(v_i, v_j) \mid cl(v_i) \neq cl(v_j) \wedge g(v_i) \neq g(v_j)\}\right|$$

denote the number of pairs of points $i < j$ that are assigned to different clusters in both cl and g (true negatives). The rand index of cl and g is then

$$R(cl, g) = \frac{TP + TN}{\binom{N}{2}} = \frac{2 \cdot (TP + TN)}{N \cdot (N - 1)}.$$

Note that the denominator is the total number of unordered pairs of points over N data points.

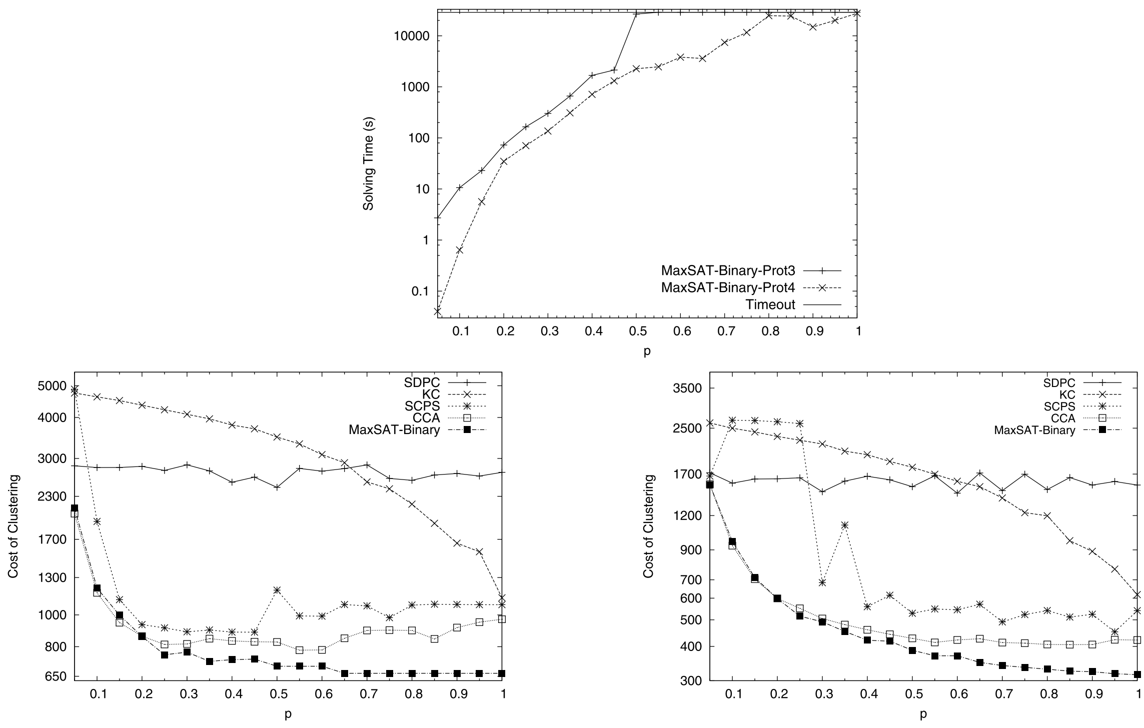

Fig. 10. Top: evolution of running times. Bottom: cost of the clusterings obtained on sparse matrices. Bottom left: Prot3, Bottom right: Prot4. For the unsolvable MaxSAT instances we report the cost of the highest value of p still solvable.

As discussed in Section 2.3, ML and CL constraints are a non-trivial addition to the correlation clustering problem. Local search style clustering algorithms tend to suffer from over-constrainment in the sense that adding too many constraints can prevent such algorithms from converging. In contrast, the running time of the MaxSAT solver decreases with added constraints, see Fig. 12.[5] As a consequence, using UK several additional datasets could be fully clustered with MaxSAT.

Finally, we demonstrate that added UK constraints steer the clusterings produced by our MaxSAT encoding effectively towards the golden clustering. Fig. 13 shows how the rand index increases as ML and CL constraints are added to the original similarity matrix. The number of extra constraints required for our algorithm to achieve rand indexes over 0.95 is for most datasets fairly small. The results suggest that extra constraints are highly beneficial and user knowledge should be taken advantage of whenever available.

9.3.2. Cluster-level constraints

To illustrate that the MaxSAT-based approach also enables obtaining optimal solutions which are guaranteed to satisfy cluster-level constraints, we consider a *Cluster Dissimilarity* constraint CL-DIS(k), closely related to constraints previously studied in distance-based clustering. Informally, a clustering cl satisfies the CL-DIS(k) constraint if no pair of points that are "more similar" than the threshold k are assigned to different clusters. More precisely, we require that $W(i, j) < k$ whenever $cl(v_i) \neq cl(v_j)$. This constraint is similar to the δ constraint in [37], where it was enforced by observing that the constraint can be decomposed into a set of ML constraints: whenever $W(i, j) > k$, we add a ML constraint over v_i and v_j.

Fig. 14 demonstrates the running time of MaxHS on the four protein datasets (without any pruning) and an added CL-DIS(k) constraint for different threshold values k. Recall that all values in the benchmark similarity matrices were normalized to be between -0.5 and 0.5, which explains why we experimented with the threshold values $0.02, 0.04, \ldots, 0.5$. Note that $k = 0.5$ means that we are solving the original instance. All in all, the results show that the running time of MaxHS decreased drastically already for values only slightly below 0.5, all such instances being solvable in under a second.

[5] Note that, especially when using a MaxSAT solver which searches bottom-up in the cost function (such as MaxHS), adding more constraints could also have a negative effect on the running times of the solver.

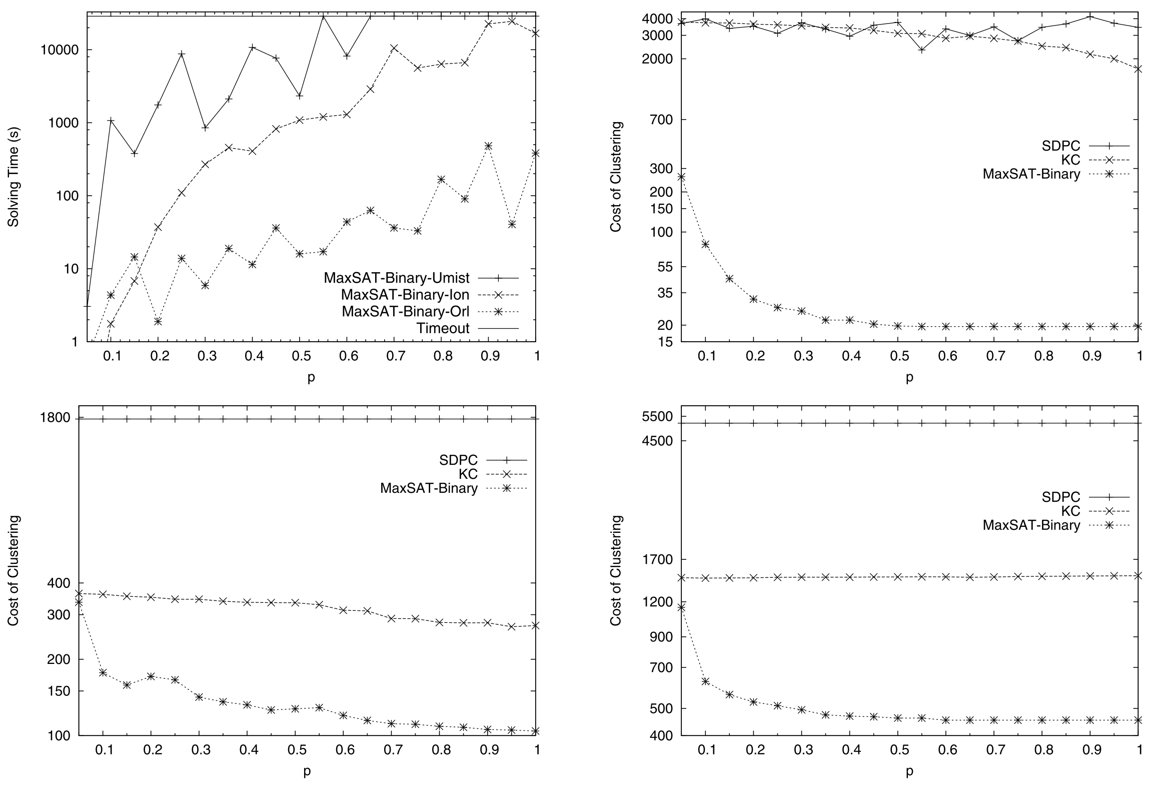

Fig. 11. Evolution of running times (top left) and the cost of the clusterings obtained on sparse matrices. Top right: orl, bottom left: ionosphere, bottom right: umist. For the unsolvable MaxSAT instances we report the cost of the highest value of p still solvable.

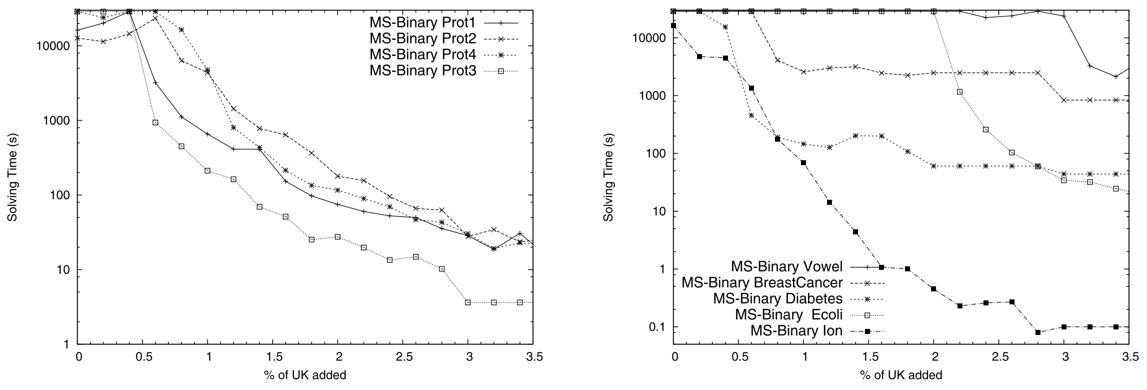

Fig. 12. Evolution of running times of the MaxSAT solver with increasing amount of user knowledge added to the matrices.

9.4. MaxSAT-specific experiments

For the rest of this section, we will focus more MaxSAT-specific questions. We will consider the effects of MaxSAT-level preprocessing and symmetry breaking, as well as the performance of different state-of-the-art MaxSAT solvers, under the best-performing binary encoding. For these experiments we used the same set of benchmarks as in Section 9.2.2. We begin by considering preprocessing.

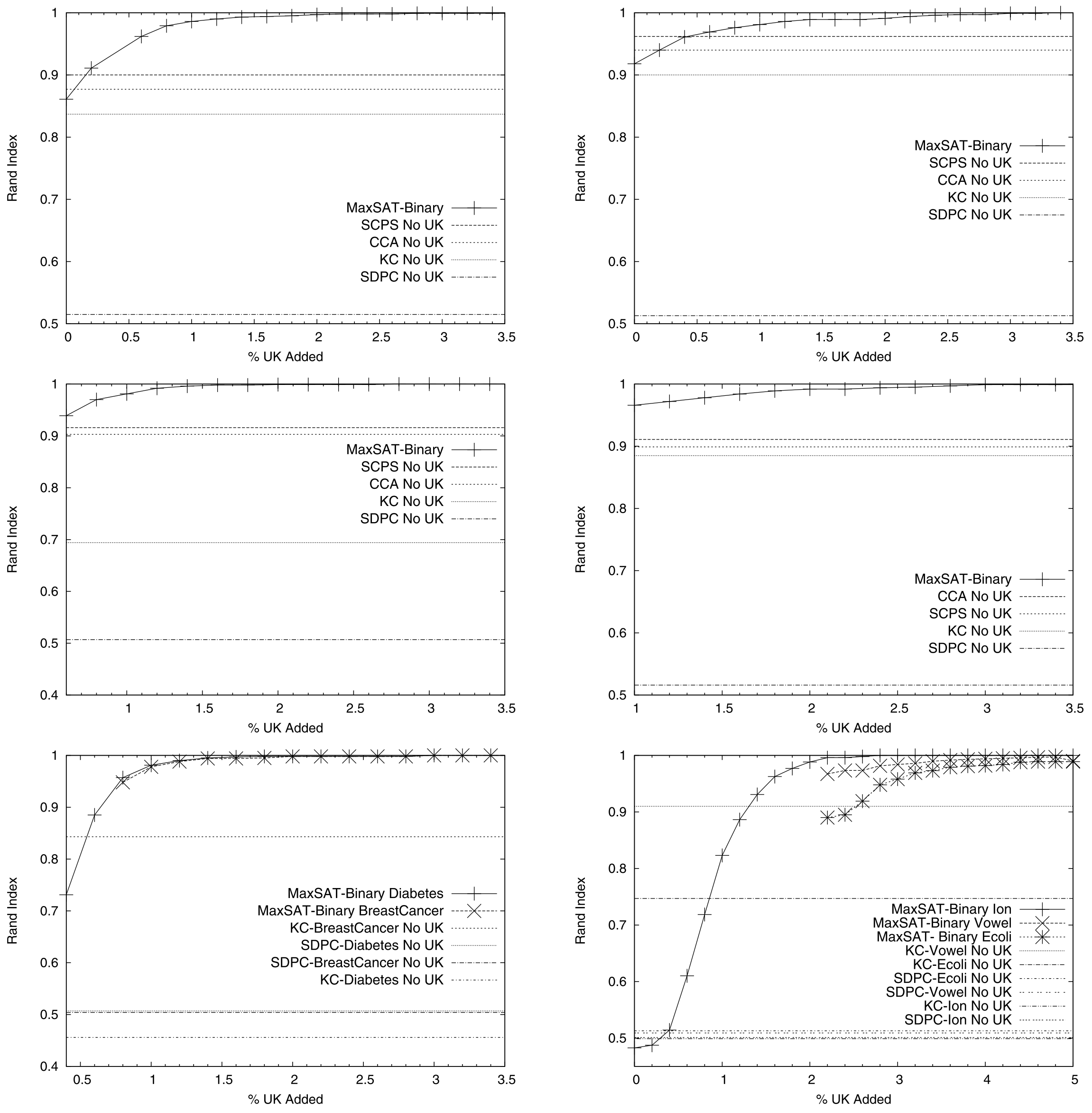

Fig. 13. Evolution of the Rand index with increasing amount of user knowledge added to the matrices. The datasets from the top, left to right are: Prot1, Prot2, Prot3, Prot4, Breastcancer/Diabetes, Ecoli/Ionosphere/Vowel.

9.4.1. Effects of MaxSAT preprocessing

Preprocessing is today an essential part of SAT solving. However, to date there has been few studies on MaxSAT preprocessing [73,74]. As such, MaxSAT preprocessing is a relatively recent area of research, possibly due to the fact that not all popular SAT preprocessing techniques can be directly applied in the context of MaxSAT [73]. However, one recently proposed way of using SAT preprocessing on MaxSAT instances is through the so-called *labeled-CNF* framework [75,73] which we also apply here.

We preprocess a given MaxSAT instance $F = (F_h, F_s, c)$ with N soft clauses in the following way.

1. Form the CNF formula $F^{SAT} = F_h \cup F_r$, where $F_r = \{(w_i \vee \neg r_i) \mid w_i \in F_s, i = 1..N\}$, with each of the variables r_i not appearing anywhere in F^{SAT} except the clause $(w_i \vee \neg r_i)$, thus obtaining the so-called *labeled-CNF* [75].
2. Apply the Coprocessor 2.0 [76] SAT preprocessor on F^{SAT}, obtaining the CNF formula F'^{SAT}. Coprocessor implements a large range of modern SAT preprocessing techniques, including unit propagation, variable elimination [77], clause

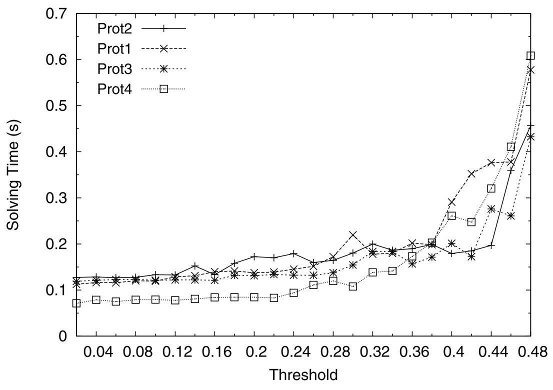

Fig. 14. Running time of the MaxSAT solver with CL-DIS(k) constraint for different values of k.

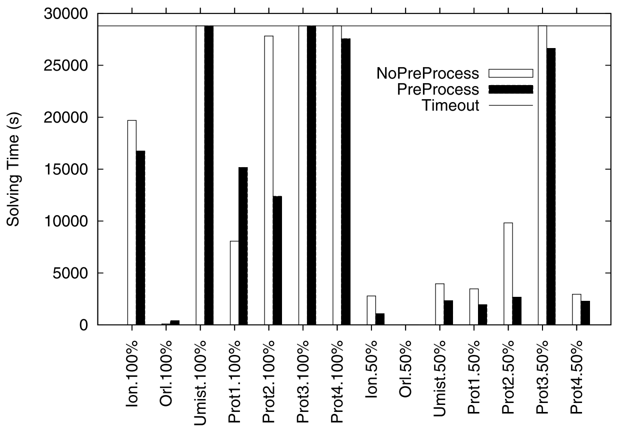

Fig. 15. Effect of preprocessing MaxSAT instances. The percentage in the instance name is the expected number of non-zero entries (compared to the full instance). Solving times reported for MaxHS and the binary encoding.

elimination [78,79], binary clause reasoning [80], etc. We used the white-listing option of Coprocessor to disable the preprocessor from removing any occurrences of any of the r_i variables. This is critical for ensuring correctness on the MaxSAT-level [75].

3. Finally, we constructed the MaxSAT instance $F^{PRE} = (F_h^{PRE}, F_s^{PRE}, c^{PRE})$ where $F_h^{PRE} = F'^{SAT}$, $F_s^{PRE} = \{(r_i) \mid i = 1..N\}$ and $c^{PRE}(r_i) = c(w_i)$. Now $\text{OPT}(F) = \text{OPT}(F^{PRE})$ and any solution τ to F^{PRE} can be extended into a solution for F of equal cost in polynomial (negligible) time, similarly as when applying SAT preprocessing on the SAT-level [81].

The preprocessing time of MaxSAT instances resulting from the binary encoding was negligible, less than 10 seconds for each instance.

Fig. 15 demonstrates the difference in running time with and without preprocessing on instances consisting of 50% and 100% of the non-zero values of the datasets. On a majority of 94 out of 140 instances, preprocessing lowered the running time of the MaxSAT solver enough to compensate for the extra time spent on preprocessing. Furthermore, all instances that were solved without preprocessing were also solved after applying preprocessing. However, we did also observe instances on which preprocessing had a negative impact on the running time, exemplified in Fig. 15 by the Prot1 dataset.

9.4.2. Effect of symmetry breaking

The solution space of correlation clustering is highly symmetric: given any clustering $cl: V \to \{1 \dots |V|\}$ of a set of data points, the cluster indices can be arbitrarily permuted without affecting the actual partitioning of the data points and hence its cost. This leads to the question of whether MaxSAT solving could be sped-up by breaking some of these symmetries on the MaxSAT-level.

Full symmetry breaking seems unlikely to be beneficial, due to the fact that a very large number of clauses—going far beyond the size of the binary encoding—would be needed. More formally, define a relation $\equiv$ over the set of possible clusterings of V by $cl \equiv cl'$ if $cl = \sigma \circ cl'$ for some permutation σ. It is simple to see that the relation $\equiv$ is an equivalence relation. Full symmetry breaking corresponds to adding constraints that forbid all but one member out of each equivalence class of $\equiv$. A straightforward approach to achieve this would require clauses that identify some representative point (e.g., the smallest) assigned to each cluster in a clustering and enforce an ordering over these points. This could be done by encoding constraints stating *If v_i is not co-clustered with v_j for any $j < i$, then $cl(v_i) > cl(v_k)$ for all $k < i$.* Such a constraint can be encoded into MaxSAT by techniques similar to the ones presented in Section 8. However, as there can be up to N clusters, this would introduce $\mathcal{O}(N^3)$ new clauses, which would evidently deteriorate the performance of a MaxSAT solver. As a related observation, note that the cubic transitive MaxSAT encoding breaks all symmetries, but does not perform as well as the binary encoding (without symmetry breaking).

Even though full symmetry breaking seems infeasible, we might still be able to boost solver performance by applying partial symmetry breaking to our encoding. Partial symmetry breaking refers to including clauses that remove (only) some symmetric solutions. As a simple example and the baseline in our experiments, we have the already mentioned the very simple *first point into the first cluster* constraint that can be enforced with $\log_2 N$ unit clauses of the form $(\neg b_1^i)$, $1 \le i \le \log_2 N$.

A more involved symmetry breaking constraint we consider is the ClustersLessThan(i, N) presented in Section 8.5. Without enforcing a limit on the number of clusters, the constraints ClustersLessThan(i, N) are not required in order for the encoding to be correct. However, including them does prune away a significant number of symmetric solutions. Furthermore, the constraints are relatively compact, in total only $\mathcal{O}(N \cdot \log_2(N))$ new clauses need to be added. In our experiments we call this type of symmetry breaking RemoveSlack.

A further form of symmetry breaking deals with symmetries induced by the possibility of empty clusters. As an instance created by the binary encoding allows (at least) N different cluster indices for every point, the placement of empty clusters can potentially introduce symmetries into the solution space. Assume for example that some optimal clustering cl contains C clusters. Then cl is equivalent to at least $\binom{N}{N-C}$ other clusterings, depending on which of the N cluster indices are empty. We can remove some of these symmetries by forcing the empty clusters to occupy indices $C + 1, \ldots N$, or, more generally, the largest (or equivalently, the smallest) available indices. We next describe the encoding of this constraint in terms of the binary encoding.

Assume that we are using a bits to represent the cluster indices in the binary encoding. We introduce new variables $E_1 \ldots E_{(2^a)}$, with the intended interpretation $\tau(E_j) = 1$ iff $c(v) \neq j$ for all $v \in V$, that is, cluster j is empty. Using these variables, the empty cluster indices are propagated with constraints of the form $E_i \to E_{i+1}$ which inductively require that the clusters of higher index than an empty cluster are also empty, and that all clusters of lower index than a non-empty cluster are also non-empty. To define the E_j variables for a given cluster j and data point v_i, let b_i^{t*} denote the literal corresponding to the value of bit t of j, i.e., b_i^t if bit t of j is 1, and $\neg b_i^t$ otherwise. Now the E_j variable can be defined as

$$E_j \leftrightarrow \bigvee_{i=1}^{N} \left(b_i^{1*} \wedge \ldots \wedge b_i^{a*} \right),$$

which can be compactly represented as clauses by introducing N new auxiliary variables $C_1^j, \ldots, C_N^j$ and defining them as $C_i^j \leftrightarrow \left(b_i^{1*} \wedge \ldots \wedge b_i^{a*} \right)$. Similar constraints are introduced for all of the E_i variables. We call this type of symmetry breaking PropagateEmpty. Compared to RemoveSlack, the PropagateEmpty constraint breaks more symmetries. In particular, symmetries broken by PropagateEmpty include all of the symmetries broken by RemoveSlack. However, PropagateEmpty is more costly in terms of encoding size. In total, the constraints introduce $\mathcal{O}(N^2 \cdot \log_2 N)$ new clauses. Recall that the total size of the binary encoding is $\mathcal{O}(E \cdot \log_2 N)$ where E is at most of order N^2, which means that enforcing the PropagateEmpty constraint might significantly increase the number of clauses on sparse instances.

Fig. 16 demonstrates the effect of the RemoveSlack constraint compared to the baseline. The PropagateEmpty constraint is missing from the figure due to the fact that, when enforcing it, no instances could be solved within the timeout. We hypothesize that the reason for this is the significant number of clauses required for encoding it. As a concrete example, the preprocessed Prot1 dataset with 100% of the non-zero values present contains 323 301 clauses without PropagateEmpty and 6 427 796 clauses when enforcing PropagateEmpty. However, as can be seen in Fig. 16, the RemoveSlack constraint actually does improve solver performance on most instances.[6]

9.4.3. A comparison of MaxSAT solvers

In the main experiments reported in this work, we used the MaxHS MaxSAT solver, which has shown very good performance especially in the "crafted" category of the recent MaxSAT Evaluations.[7] Here we report on the performance of other state-of-the-art MaxSAT solvers, using the following solvers.

- Eva500 solver [82], obtained from http://www.maxsat.udl.cat/14/solvers/.
- MsUnCore bcd2 version [49,48] obtained from http://www.csi.ucd.ie/staff/jpms/soft/soft.php.

[6] We remind the reader that, apart from the *first point into the first cluster* constraint, symmetry breaking was not applied in the other experiments reported on in this article.

[7] http://www.maxsat.udl.cat/14/results/index.html#wpms-crafted.

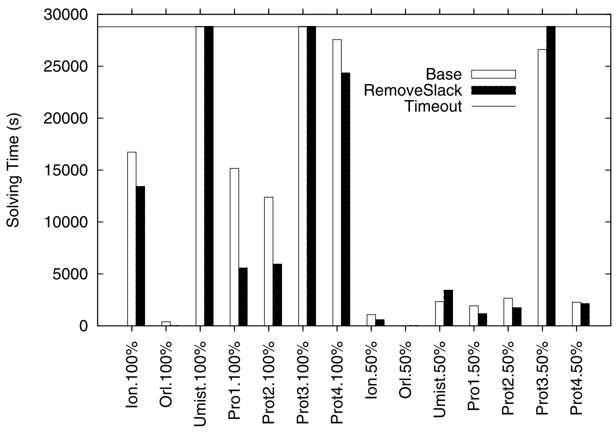

Fig. 16. Effect of different symmetry breaking techniques on the solving time of MaxSAT. The percentage in the instance name is the expected number of non-zero entries (compared to the full instance). Solving times reported for MaxHS and the binary encoding.

Table 2
Comparison of MaxSAT solvers.

Solver	Number of solved instances	Number of timeouts
MaxHs	121	19
Eva500	65	75
ILP2013	7	133
MsUnCore	7	133
OpenWBO	0	140

- OpenWBO [83,84] version 1.1.1. obtained from http://sat.inesc-id.pt/open-wbo/.
- The ILP2013 solver [56]. We implemented the conversion to an integer program ourselves and used CPLEX to solve the resulting instances.

The first three in the list are core guided solvers (recall Section 5.2). Eva500 uses the identified cores and a restricted form of MaxSAT resolution [85] to relax the MaxSAT instance in a controlled way. MsUnCore performs binary search over the cost function and also maintains a set of already identified disjoint cores and relaxes each core separately whenever a new one is found. OpenWBO uses an incremental approach that allows it to pertain the state of the internal SAT solver more efficiently between the iterations. ILP2013 encodes the whole MaxSAT instances as an integer linear program and then calls an ILP solver. Since MsUnCore, OpenWBO and Eva500 accept only integral weights, for running these solvers we multiplied all similarity values by 10^{13} (the highest possible multiplier with which the trivial cost upper bound required as input by the solvers still stays within the 2^{63} range) and rounded afterwards to integers. Table 2 gives a performance comparison of the solvers. MaxHS scales significantly better than the other solvers. All in all, MaxHS solved 121 instances within the timeout while the second-best performing Eva500 solved 65. The other solvers in the comparison timed out on most instances. Note that, apart from the *first point into the first cluster* constraint, symmetry breaking was not applied in this experiment.

10. Related work

We continue with a survey on related work.

10.1. Correlation clustering

While the notion of producing good clusterings under inconsistent advice first appeared in [11], the formal definition of correlation clustering was proposed in [4] and shown to be NP-hard on complete graphs with each edge labeled with $+$ or $-$; or, in terms of the general problem definition considered in this work, on symmetric similarity matrices W where $W(i,j) = \{-1, 1\}$ for all i and j. NP-hardness motivated early work on approximative algorithms for the problem. Approximation algorithms for correlation clustering typically address one of three different objectives for the problem: *minimizing disagreements*, *maximizing agreements*, or *maximizing correlation*. Given a similarity matrix W over a set of data points

$V = \{v_1, \ldots, v_N\}$, minimizing disagreements refers to minimizing the number of point pairs v_i, v_j whose cluster assignment does not agree with their similarity value $W(i,j)$, or more precisely, to finding a clustering *cl* minimizing

$$\sum_{\substack{i<j \\ \infty > W(i,j) > 0}} \mathcal{I}[cl(v_i) \neq cl(v_j)] W(i,j) + \sum_{\substack{i<j \\ -\infty < W(i,j) < 0}} \mathcal{I}[cl(v_i) = cl(v_j)] |W(i,j)|. \tag{7}$$

Maximizing agreements refers to maximizing the number of pairs of points v_i, v_j whose cluster assignment agrees with their similarity value $W(i,j)$, or more precisely, to finding a clustering *cl* maximizing

$$\sum_{\substack{i<j \\ \infty > W(i,j) > 0}} \mathcal{I}[cl(v_i) = cl(v_j)] W(i,j) + \sum_{\substack{i<j \\ -\infty < W(i,j) < 0}} \mathcal{I}[cl(v_i) \neq cl(v_j)] |W(i,j)|. \tag{8}$$

Maximizing correlation refers to maximizing the difference between agreements and disagreements, i.e., using the objective function obtained by substracting Equation (7) from Equation (8).

A polynomial-time approximation scheme for maximizing agreements on complete symmetric matrices with $\{-1,1\}$ similarity values was presented in [4]. No such scheme is likely to exist for minimizing disagreements as the problem is APX-hard [86]. On general matrices, maximizing agreements is also APX-hard [6], except for when the ratio between the smallest and largest absolute value in the matrix is bounded by a constant [33]. To the best of our knowledge, the SDPC algorithm proposed in [34] and detailed below is the best known approximation algorithm for maximizing correlation.

The two approximative algorithms for correlation clustering we experimented with in this work are based on different techniques. KwickCluster [5] is a greedy combinatorial approximation algorithm that at each iteration picks one of the still unassigned nodes to be the *pivot node*, and forms a new cluster containing the pivot node and all still unassigned nodes that are similar to the pivot node (recall that nodes v_i and v_j are similar if $W(i,j) > 0$ in the similarity matrix under consideration). The algorithm terminates when all points have been assigned to some cluster. The same algorithm also appears in [67] under the name *PivotAlg*. As shown in [5,67], KwickCluster is a factor-3 approximation algorithm for minimizing disagreements under the assumption that $W(i,j) \in \{-1,1\}$ for all i and j, and a factor-5 approximation algorithm under the assumption that $-1 \leq W(i,j) \leq 1$ for all i, j.[8]

SDPC [34] is based on rounding solutions to a semi-definite program that itself is a relaxation of the quadratic programming formulation of correlation clustering restricted to two clusters. Restricting the correlation clustering problem search space to clusterings only containing two clusters, the quadratic program in Equation (4) (recall Section 4) can be formulated equivalently as

$$\begin{array}{ll} \textsc{Maximize} & \displaystyle\sum_{i=1}^{N} \sum_{j=i+1}^{N} \big(W(i,j) z_i z_j\big) \\ \text{subject to:} & z_i \in \{-1,1\} \text{ for all } i, \end{array} \tag{9}$$

where N is the number of data points, and the value in the solution of the variable z_i indicates whether point v_i is assigned to cluster 1 or -1. This quadratic program can be relaxed into the semi-definite program

$$\begin{array}{ll} \textsc{Maximize} & \displaystyle\sum_{i=1}^{N} \sum_{j=i+1}^{N} \big(W(i,j) u_i \cdot u_j\big) \\ \text{subject to:} & |u_i| = 1 \text{ for all } i \\ & u_i \in \mathbb{R}^N \text{ for all } i, \end{array} \tag{10}$$

where each z_i binary variable from Equation (9) is represented by a vector u_i on the unit sphere in $\mathbb{R}^N$. The relaxation of the quadratic program (Equation (9)) into the semi-definite program (Equation (10)) is standard, being similar to the SDP relaxation for MaxCut presented in [87].

In [34] an algorithm that rounds a solution obtained to Equation (10) into a well-defined clustering is presented and shown to achieve an $\Omega\left(\log(N)^{-1}\right)$ approximation factor for maximizing correlation. The algorithm compares clusterings obtained from rounding the semi-definite program with the (unique) cost of the trivial clustering in which all data points are assigned to different clusters, and returns the better solution out of these two.

In [33] the authors develop a PTAS for maximizing agreements on general matrices under the assumption that the ratios between weights in the input matrices are bounded by a constant, which implies that the matrices cannot contain 0-entries, i.e. the available similarity information has to be complete. The PTAS is developed by using the smooth polynomial programming technique on the QIP formulation, which results in strong approximation bounds for the maximization problem on matrices satisfying the assumption.

[8] Recall that a factor α approximation algorithm on a minimization (maximization) problem is guaranteed to return a solution of cost lower (higher) than α times the cost of the optimal solution.

In [88] a more restricted version of the approximative correlation clustering algorithm of [4] is presented, with experiments on identifying and resolving noun co-reference in texts. In [89] a greedy randomized adaptive search procedure (GRASP) based approximative algorithm is presented, with the motivation that the obtained solutions can be used as a criterion for determining the balance in social networks. Also, in [10] correlation clustering is used for crosslingual link detection between google news groups. The authors build on the results of [86] and present an algorithm based on relaxing the ILP formulation of correlation clustering into a linear program and then using region growing techniques for rounding of the solution of the linear program. As such, their algorithm is also approximative in nature and, in contrast to our approach, cannot provide optimality guarantees. The authors also provide some results on exactly solving the ILP with added must-link and cannot-link constraints.[9] As noted in [10] and supported by our experiments, the ILP-based approach to correlation clustering suffers from the fact that the number of constraints is cubic in the number of data points, leading to memory problems in practice. The authors of [10] approach the issue by splitting the LP into smaller chunks and processing the chunks separately. In contrast, our experimental results suggest that using MaxSAT for solving correlation clustering is more memory-efficient without extra tuning. There has also been some work done on a variant of correlation clustering in which the search is further restricted to $cl\colon V \to \{1, \ldots K\}$ for some $K < N$ [8]. As explained in Sections 7 and 8, both the binary and the unary encoding can be used in this setting as well.

A few generalizations of correlation clustering have been proposed. In [12] the authors experiment with correlation clustering allowing overlapping clusters. The proposed solution to overlapping correlation clustering is a local search algorithm that locally adjusts the solution clustering as long as the cost function decreases. Out of the MaxSAT encodings presented in this work, the unary encoding extends naturally to overlapping clustering by changing the cardinality constraint $\textsc{ExactlyOne}(i)$ of each point to a more general $\sum_{k=1}^{K} y_i^k \leq p$, where p is the maximum number of clusters a single point can be assigned to. The resulting encoding can be shown to produce globally optimal solutions to the overlapping clustering problem. Another proposed generalization to correlation clustering is chromatic correlation clustering [13]. In the basic form of correlation clustering, there are two possible relationships between pairs of data points. A pair of data points can either be similar, dissimilar (or neither). Chromatic correlation clustering generalizes this by allowing more than two different categories of relationships. This can be visualized as an undirected graph in which each edge is colored. The task is then to find a clustering that maximizes color purity of edges within clusters. Our MaxSAT encodings can be extended to cover Chromatic Correlation Clustering by introducing variables which represent the principal color of each cluster.

10.2. Constrained clustering

As exemplified in Section 9.3, the MaxSAT-based approach allows for obtaining solution which are guaranteed to satisfy additional hard constraints on the clusterings of interest. This includes both instance-level and, as exemplified in Section 9.3.2, even some distance-based cluster-level constraints which have been previously studied in the context of constrained clustering [90,23]. The idea of adding constraints to the clustering problem was first introduced in [27,28]. The introduction of constraints to the clustering problem allows the addition of domain knowledge to the problem and has been shown to increase clustering accuracy [27]. Much of the early work on constrained clustering concentrated on modifying existing heuristics and clustering algorithms in order to allow the addition of constraints. Examples include k-means and COB-WEB [27,28], EM [91], hierarchical [92] and spectral clustering [93]. The problem of deciding if there exists a clustering satisfying a given set of must and cannot-link constraints was shown to be NP hard in [37]. In fact, many of the modified approximative algorithms are not even guaranteed to return a clustering satisfying all user constraints. The algorithms also have difficulties in handling too many extra constraints, they are easily over-constrained, preventing the algorithms from converging at all [37].

An alternative approach to constrained clustering is to cast the task as a constraint optimization problem, allowing for a very natural incorporation of added constraints. This is the approach which we employ in this work. A similar idea was proposed in [23] in a different clustering setting. The authors show that a satisfiability-based framework is well-suited for constrained clustering in the sense that constraints are easily added, the solutions returned are guaranteed to be globally optimal and satisfy all given constraints, and the search algorithm is not as easily over-constrained. Our approach to solving constrained *correlation* clustering is similar, but more generic as we do not restrict ourselves to only allowing two distinct clusters, which is a polynomial time special case of the general clustering problem. Furthermore, our encoding are on the MaxSAT-level (optimization instead of pure SAT), and employ a MaxSAT solver instead of a pure SAT solver.

Constrained clustering has also been approached via integer programming. In [94,95] a variety of different possible constraints and optimization functions are considered. However, in practice their approach might be difficult to use as it requires a predetermined set of candidate clusters from which the algorithm searches for the best subset. In [90,96] the authors use an integer programming and column generation based approach in order to exactly solve the minimum sum of squares clustering problem. Constrained clustering has also been approached, again in a different clustering setting, by constraint programming (CP) [97,24].[10] In [97] different optimization criteria for clustering are studied and solved by casting the clustering problems as constraint programming problems. A SAT-based framework of constrained clustering has

[9] Unfortunately, the authors were unable to provide an implementation of their algorithm.

[10] The term constraint programming refers here to the declarative language as opposed to a general term of the paradigm.

also been proposed for example in [31], but optimization criteria are not applied in their experiments. In [24] a general framework for *K-pattern set mining under constraints* is introduced. The authors present a general framework and explore the strengths and limitations of using constraint programming. Yet another recent example of using declarative programming in the context of clustering is [26], in which an ILP formulation of hierarchical clustering, with an explicit objective function that is globally optimized, was presented; that approach would similarly allow for satisfying hard constraints over the solution space.

11. Conclusions

This work contributes to the research direction of harnessing constraint solving for developing novel types of generic data analysis techniques. The focus of our study is the applicability of state-of-the-art Boolean optimization procedures to cost-optimal correlation clustering in both unconstrained and constrained settings. To this end, we presented a novel MaxSAT-based framework for solving correlation clustering. Our approach is based on casting the clustering problem declaratively as weighted partial maximum satisfiability, and using a generic MaxSAT solver for finding cost-optimal clusterings. We studied three different encodings of correlation clustering as MaxSAT, and reported on an experimental evaluation, comparing both the time required to solve the resulting MaxSAT instances, and the quality of the clusterings obtained. We compared the MaxSAT-based approach to previously proposed both exact (integer linear and quadratic programming based) and approximative (specialized local search and approximation algorithms and semi-definite programming) approaches on real-world datasets. The MaxSAT approach scales better than the exact integer linear and quadratic programming approaches, and provides clusterings of significantly lower cost than the approximative algorithms, especially when the input data is sparse. Due to the intrinsic computational hardness of correlation clustering, we acknowledge that a potential issue with our approach is scalability, especially scaling the MaxSAT-based approach to very large datasets (with tens of thousands of data points). Nevertheless, the approach can provide cost-optimal clusterings on real-world datasets with close to a thousand points. The approach is also flexible when it comes to satisfying user-specified constraints, i.e., in constrained correlation clustering. The running times of the approach can notably decrease in a constrained setting, allowing for solving larger datasets faster compared to the non-constrained setting. This is in stark contrast with local search algorithms which easily suffer from over-constraining in constrained settings. It is conceivable that our approach can be improved also by foreseeable improvements to generic MaxSAT solvers and by developing domain-specific parallelization schemes, as well as by specialized constraint optimization techniques and heuristics for the problem domain. Yet another interesting direction would be to study the applicability of Large Neighborhood Search which combine local search strategies for fixing a subspace of the search space to which to apply exact search techniques.

Appendix A. Proofs

We provide detailed proofs of the fact that any similarity matrix can be symmetrized without affecting the set of optimal clusterings, as discussed in Section 2.2, and the correctness of the three encodings of correlation clustering as MaxSAT, presented in Sections 6, 7 and 8.

A.1. Proof of Theorem 1

Assume that $V = \{v_1 \dots v_N\}$ is a set of N data points and $W \in \overline{\mathbb{R}}^{N\times N}$ is an asymmetric similarity matrix. Let H' be the non-simplified cost function of correlation clustering (Equation (2)) and H the simplified cost function (Equation (1)). We will assume w.l.o.g. that none of the considered matrices include contradicting infinite values.

The proof of Theorem 1 consists of considering the two different possible sources of asymmetries. The first are pairs of indices i and j for which $W(i, j) < 0 < W(j, i)$. Any such pair will always incur a cost of at least $\min(|W(i, j)|, W(j, i))$ to any clustering. Thus the absolute value of both $W(i, j)$ and $W(j, i)$ can be decreased by this minimum without affecting the set of optimal clusterings. Notice that after this either $W(i, j) = 0$ or $W(j, i) = 0$. This observation is formalized in Lemma 1.

Based on the above, we can assume that all pairs $W(i, j)$ and $W(j, i)$ have the same sign. Now the existence of the symmetric W^S follows from the following observations. If both $W(i, j)$ and $W(j, i)$ are non-positive, the points v_i and v_j either incur a cost of $|W(i, j)| + |W(j, i)|$ or 0 to $H'(W, cl)$ under any clustering *cl*. Analogously, if both are non-negative, then the points either incur a cost of $W(i, j) + W(j, i)$ or 0. Hence, by letting $W^S(i, j) = W(i, j) + W(j, i)$, *cl* will incur the same cost under W^S (as measured by H) as under W (as measured by H'). This discussion is formalized in the proof of Theorem 1 given after the proof of Lemma 1.

Lemma 1. *There is a similarity matrix W^T such that $W^T(i, j) \cdot W^T(j, i) \geq 0$ (i.e., both have the same sign) for all i and j and* $\mathrm{argmin}_{cl}(H'(W, cl)) = \mathrm{argmin}_{cl}(H'(W^T, cl))$.

Proof. W^T can be constructed by repeatedly applying Lemma 2 to each pair of indices corresponding to elements of opposing signs in W. □

Lemma 2. *Let i and j be any pair of indices for which $W(i,j) < 0 < W(j,i)$. There exists a similarity matrix W^t for which $W^t(i,j) \cdot W^t(j,i) = 0$ and $\text{argmin}_{cl}(H'(W,cl)) = \text{argmin}_{cl}(H'(W^t,cl))$.*

Proof. Construct W^t as

$$W^t(i,j) = W(i,j) + \min(|W(i,j)|, W(j,i)),$$
$$W^t(j,i) = W(j,i) - \min(|W(i,j)|, W(j,i)) \text{ and}$$
$$W^t(i',j') = W(i',j') \text{ whenever } i \neq i' \text{ or } j \neq j'.$$

Now either $W^t(i,j) = 0$ or $W^t(j,i) = 0$, and hence $W^t(i,j) \cdot W^t(j,i) = 0$. Notice also that W^t includes exactly the same infinite values as W. This means that the set of feasible clusterings is the same for both matrices. We prove the second part of the lemma by showing that

$$H'(W,cl) = H'(W^t,cl) + \min(|W(i,j)|, W(j,i)) \tag{A.1}$$

for any feasible clustering cl of V. The fact that the set of optimal clusterings under W is the same as under W^t follows from $\min(|W(i,j)|, W(j,i))$ being independent of cl.

First, if either $W(i,j)$ or $W(j,i)$ is infinite, then it is infinite in W^t. Furthermore, the other element is 0 in W^t. Hence the pair i, j will incur cost $\min(|W(i,j)|, W(j,i))$ under W and 0 under W^t. As all other elements are equal in both matrices, we have $H'(W,cl) = H'(W^t,cl) + \min(|W(i,j)|, W(j,i))$.

Assume now that both $W(i,j)$ and $W(j,i)$ are finite. As the transformation from W to W^t maintains signs of all elements, Equation (A.1) is equivalent to

$$\begin{aligned}&\mathcal{I}[cl(v_i) = cl(v_j)] \cdot |W(i,j)| + \mathcal{I}[cl(v_j) \neq cl(v_i)] \cdot W(j,i)\\ &= \mathcal{I}[cl(v_i) = cl(v_j)] \cdot |W^t(i,j)| + \mathcal{I}[cl(v_j) \neq cl(v_i)] \cdot W^t(j,i) + \min(|W(i,j)|, W(j,i)).\end{aligned}$$

This can be verified by considering the possible cases separately. □

Proof of Theorem 1. By Lemma 1 we can assume that $W(i,j) \cdot W(j,i) \geq 0$ for all i and j. Let $W^S(i,j) = W(i,j) + W(j,i)$. It is clear that W^S is symmetric. It remains to be shown that $\text{argmin}_{cl}(H(W^S,cl)) = \text{argmin}_{cl}(H'(W,cl))$. First note that $W^S(i,j) = \pm\infty$ iff either $W(i,j) = \pm\infty$ or $W(j,i) = \pm\infty$, so the set of feasible clusterings is the same for both matrices.

Let $i < j$ and cl be any feasible clustering of V. We show that $H(W^S,cl) = H'(W,cl)$. By decomposing both H and H' as in the proof of Lemma 1, is enough to show that

$$\begin{aligned}&\mathcal{I}[-\infty < W(i,j) < 0] \cdot |W(i,j)| + \mathcal{I}[-\infty < W(j,i) < 0] \cdot |W(j,i)|\\ &= \mathcal{I}[-\infty < W^S(i,j) < 0] \cdot |W^S(i,j)|\end{aligned}$$

and

$$\begin{aligned}&\mathcal{I}[\infty > W(i,j) > 0] \cdot W(i,j) + \mathcal{I}[\infty > W(j,i) > 0] \cdot W(j,i)\\ &= \mathcal{I}[\infty > W^S(i,j) > 0] \cdot W^S(i,j),\end{aligned}$$

corresponding to the two possible scenarios, $cl(v_i) = cl(v_j)$ and $cl(v_i) \neq cl(v_j)$, respectively. Both equations follow from the fact that the transformation from W to W^S preserves the signs of all elements. Thus $|W^S(i,j)| = |W(i,j) + W(j,i)|$. □

A.2. Correctness of the MaxSAT encodings

Next we move on to prove the correctness of the three MaxSAT encodings presented in this work, in other words, we prove Theorems 2, 3 and 4. Again, let $V = \{v_1, \dots v_N\}$ be a set of data points, $W \in \overline{\mathbb{R}}^{N \times N}$ a symmetric similarity matrix, and K an upper bound on the available clusters. Note that we allow $K = N$, so the proofs presented here cover the problem definition of [4,12] and [9] as well as [8]. We first consider general conditions for correct MaxSAT encodings of correlation clustering. Recall that H is the cost function, Equation (1), of correlation clustering under minimization.

Proposition 1. *Let F be a MaxSAT instance and assume that a clustering $cl_\tau : V \to \{1 \dots K\}$ can be constructed from any solution τ to F. Further assume the following.*

1. *cl_τ is well-defined for all solutions τ to F.*
2. *For each solution τ to F, cl_τ respects the infinite values of W.*
3. *For each clustering cl that respects the infinite values of W, there exists some solution τ to F for which $H(W,cl) = H(W,cl_\tau)$.*
4. *$\text{COST}(F,\tau) = H(W,cl_\tau)$ for any solution τ to F.*

Now, if τ^* is an optimal solution to F, then cl_{τ^*} is an optimal clustering of V.

Proof. First note that Condition 1 ensures that cl_{τ^*} is well-defined and Condition 2 ensures that cl_{τ^*} is indeed a solution to the constrained problem. Now let cl be any clustering that respects the infinite values of W. Then by Condition 3 there exists a solution τ to F such that $H(W, cl_\tau) = H(W, cl)$. By the optimality of τ^* and condition Condition 4 it follows that

$$H(W, cl) = H(W, cl_\tau) = \textsc{cost}(F, \tau) \geq \textsc{cost}(F, \tau^*) = H(W, cl_{\tau^*}),$$

and hence cl_{τ^*} is optimal. □

Next we prove Theorems 2, 3 and 4 by showing that the instances generated with the transitive, unary and binary encodings fulfill the assumptions of Proposition 1.

A.2.1. Correctness of the transitive encoding

Let $F^1 = (F^1_h, F^1_s, c)$ be a MaxSAT instance generated by the transitive encoding, and cl_τ be the clustering constructed from a solution τ to F^1 by the procedure described in Section 6.4. The proof of Theorem 2, i.e., the fact that the transitive encoding produces optimal clusterings, follows from the following lemmas.

Lemma 3. *For any solution τ to F^1 and any $i < j$, we have $\tau(x_{ij}) = 1 \Leftrightarrow cl_\tau(v_i) = cl_\tau(v_j)$.*

Proof. Assume $cl_\tau(v_i) = k$. The lemma follows from the two possible scenarios that can occur when constructing cl_τ at iteration k.

(i) i is the smallest not yet assigned index. Then clearly $\tau(x_{ij}) = 1 \Leftrightarrow cl_\tau(v_i) = k = cl_\tau(v_j)$.

(ii) Some other index $t < i$ for which $\tau(x_{ti}) = 1$ is the smallest non-assigned index. Now $\tau(x_{ij}) = 1 \Leftrightarrow \tau(x_{tj}) = 1 \Leftrightarrow cl_\tau(v_i) = k = cl_\tau(v_j)$. The first equivalence follows from τ being a solution to F^1. Thus $\tau((\neg x_{ij} \vee \neg x_{ti} \vee x_{tj})) = 1$, implying $\tau(x_{ij}) = 1 \Rightarrow \tau(x_{tj}) = 1$, and $\tau((\neg x_{ti} \vee \neg x_{tj} \vee x_{ij})) = 1$, implying $\tau(x_{ij}) = 0 \Rightarrow \tau(x_{tj}) = 0$. □

Lemma 4 *(Condition 1 of Proposition 1). cl_τ is well-defined for all solutions τ to F^1.*

Proof. Trivial, as each point is assigned to at most one cluster by the procedure in Section 6.4 and the procedure only terminates after all points have been assigned to a cluster. □

Lemma 5 *(Condition 2 of Proposition 1). cl_τ respects the infinite values of W for all solutions τ to F^1.*

Proof. First notice that, due to the hard unit clauses (x_{ij}) and $(\neg x_{ij})$, $\tau(x_{ij}) = 1$ for all $W(i, j) = \infty$, and $\tau(x_{ij}) = 0$ for all $W(i, j) = -\infty$. The rest follows from Lemma 3. □

Lemma 6 *(Condition 3 of Proposition 1). For each clustering cl that respects the infinite values of W there exists some solution τ to F for which $H(W, cl) = H(W, cl_\tau)$.*

Proof. We construct such a τ as follows:

$$\tau(x_{ij}) = \begin{cases} 1 \text{ if } cl(v_i) = cl(v_j) \\ 0 \text{ else.} \end{cases}$$

Notice that τ satisfies all hard transitivity clauses since cl is well-defined. Furthermore, τ satisfies all unit hard clauses since cl respects the infinite values of W. Finally, the claim $H(W, cl) = H(W, cl_\tau)$ follows from Lemma 3 and the construction of τ as $cl(v_i) = cl(v_j) \Leftrightarrow \tau(x_{ij}) = 1 \Leftrightarrow cl_\tau(v_i) = cl_\tau(v_j)$. □

Lemma 7 *(Condition 4 of Proposition 1). $\textsc{cost}(F^1, \tau) = H(W, cl_\tau)$ holds for any solution τ to F^1.*

Proof. We consider the part $H(W, cl_\tau) \leq \textsc{cost}(F^1, \tau)$. The other direction is almost identical. A similar pair of points v_i and v_j incurs a cost $W(i, j)$ to $H(W, cl_\tau)$ iff $cl_\tau(v_i) \neq cl_\tau(v_j)$. By Lemma 3, $\tau(x_{ij}) = 0$, and hence τ does not satisfy the unit soft clause (x_{ij}) of weight $W(i, j)$. Similarly, a dissimilar pair of points v_i, v_j incurring a cost $W(i, j)$ to $H(W, cl_\tau)$ corresponds to one unsatisfied soft clause $(\neg x_{ij})$ of the same weight. □

A.2.2. Correctness of the unary encoding

Let F^2 be a MaxSAT instance generated with the unary encoding and, given a solution τ to F^2, let cl_τ be the clustering constructed form τ by the procedure described in Section 7.5. The proof of Theorem 3 follows from the following lemmas.

Lemma 8 *(Condition 1 of Proposition 1). cl_τ is a well-defined clustering.*

Proof. Follows directly from the fact that, for any point v_i, $\tau(\textsc{ExactlyOne}(i)) = 1$, and hence there exists exactly one $1 \le k \le K$ for which $\tau(y_i^k) = 1$. □

Lemma 9 *(Condition 2 of Proposition 1). cl_τ respects the infinite values of W for all solutions τ to F^2.*

Proof. Let v_i be an arbitrary data point. Assume $cl_\tau(v_i) = k$. It follows that $\tau(y_i^k) = 1$. The hard clause $(\neg y_i^k \lor y_j^k)$ for each j s.t. $W(i, j) = \infty$ implies $\tau(y_j^k) = 1$ and $cl_\tau(v_j) = k = cl_\tau(v_i)$. The hard clause $(\neg y_i^k \lor \neg y_j^k)$ for each j s.t. $W(i, j) = -\infty$ implies $\tau(y_j^k) = 0$ and $cl_\tau(v_j) \neq k = cl_\tau(v_i)$. □

Lemma 10 *(Condition 3 of Proposition 1). Let $cl\colon V \to \{1, 2, \ldots, K\}$ be any clustering of V that respects the infinite values of W. There is a solution τ to F^2 such that $cl = cl_\tau$.*

Proof. We construct such a τ. For each $1 \le i \le N$ and $1 \le k \le K$, let

$$\tau(y_i^k) = \begin{cases} 1 \text{ if } cl(v_i) = k \\ 0 \text{ else,} \end{cases} \quad \tau(A_{ij}^k) = \begin{cases} 1 \text{ if } cl(v_i) = cl(v_j) = k \\ 0 \text{ else} \end{cases} \text{ and}$$

$$\tau(D_{ij}) = \begin{cases} 1 \text{ if } cl(v_i) = cl(v_j) \\ 0 \text{ else.} \end{cases}$$

Clearly $cl = cl_\tau$ as long as τ is a solution to F^2. We show that it is by considering the different types of hard constraints present in F^2.

1. Since cl is well-defined, there is exactly one k for which $cl(v_i) = k$ for each v_i. Hence $\tau(\textsc{ExactlyOne}(i)) = 1$ for all $v_i \in V$.
2. By construction $\tau(A_{ij}^k) = \tau(y_i^k \land y_j^k)$ for all similar v_i, v_j and k. Hence $\tau(\textsc{HardSimilar}(i, j, k)) = 1$.
3. If $\tau(y_i^k) = 0$ or $\tau(y_j^k) = 0$ for a dissimilar pair of points v_i, v_j, then $\tau(\neg y_i^k \lor \neg y_j^k \lor D_{ij}) = 1$. If $\tau(y_i^k) = \tau(y_j^k) = 1$, then $cl(v_i) = cl(v_j)$. Hence $\tau(D_{ij}) = 1$ and $\tau(\neg y_i^k \lor \neg y_j^k \lor D_{ij}) = 1$. Thus $\tau(\textsc{HardDissimilar}(i, j, k)) = 1$ for all dissimilar v_i, v_j and k.
4. For all $W(i, j) = \infty$, we have that $cl(v_i) = cl(v_j)$. Hence there exists a k for which $\tau(y_i^k) = \tau(y_j^k) = 1$. Since $\tau(\textsc{ExactlyOne}(v_i)) = \tau(\textsc{ExactlyOne}(v_j)) = 1$, $\tau(y_i^{k'}) = \tau(y_j^{k'}) = 0$ for all other k'. Hence $\tau(y_i^k \leftrightarrow y_j^k) = 1$ holds for all k and $\tau(\mathrm{ML}^U(v_i, v_j)) = 1$.
5. For all $W(i, j) = -\infty$ we have that $cl(v_i) \neq cl(v_j)$. Hence either $cl(v_i) \neq k$ or $cl(v_j) \neq k$ for all k. By the construction of τ it follows that $\tau(\neg y_i^k \lor \neg y_j^k) = 1$ and $\tau(\mathrm{CL}^U(v_i, v_j)) = 1$. □

Lemma 11 *(Condition 4 of Proposition 1). $\textsc{cost}(F^2, \tau) = H(W, cl_\tau)$ for any solution τ to F^2.*

Proof. We consider the part $H(W, cl_\tau) \le \textsc{cost}(F^2, \tau)$. The other direction is almost identical. A similar pair of points v_i, v_j incurs a cost $W(i, j)$ to $H(W, cl_\tau)$ iff $cl_\tau(v_i) \neq cl_\tau(v_j)$. Either $\tau(y_i^k) = 0$ or $\tau(y_j^k) = 0$ (or both) for all k, and hence $\tau(A_{ij}^k) = 0$ for all k. Thus τ does not satisfy the soft clause $\textsc{SoftSimilar}(i, j)$ with weight $W(i, j)$. Similarly a dissimilar pair of points v_i v_j incurs cost $|W(i, j)|$ to $H(W, cl_\tau)$ iff $cl_\tau(v_i) = cl_\tau(v_j)$. There is a k for which $\tau(y_i^k \land y_j^k) = 1$. Thus τ does not satisfy the unit soft clause $(\neg D_{ij})$ with weight $|W(i, j)|$. □

A.2.3. Correctness of the binary encoding

Let F^3 be a MaxSAT instance generated with the binary encoding and, given a solution τ to F^3, let cl_τ be the clustering constructed from τ by the procedure described in Section 8.4. We prove the correctness of the binary encoding for an arbitrary K. Let $k = \lceil \log_2 K \rceil$ and assume that the encoding contains k bit variables for each data point. For any number $a \in \mathbb{N}$, let a^n denote the nth bit in the bit representation of a. For any set of bits $b^k, \ldots, b^1$, denote by $(b^k \ldots b^1)_2$ the value of the bit vector interpreted as a binary number, least significant bit to the right. Finally, let $b_i^{n*} = b_i^n$ if $K^n = 1$ and $b_i^{n*} = \neg b_i^n$ if $K^n = 0$. The proof of Theorem 4, i.e., of the fact that the binary encoding produces optimal clusterings, follows from the following lemmas.

Lemma 12 *(Condition 1 of Proposition 1). cl_τ is a well-defined clustering.*

Proof. Follows from the fact that for any point v_i, τ has to assign all the values $\tau(b_i^k), \ldots, \tau(b_i^1)$ in some unique way. Hence the value $cl_\tau(v_i)$ is uniquely defined. What remains to be shown is that $1 \le cl_\tau(v_i) \le K$. Assume for contradiction that $cl_\tau(v_i) = A$ for some $A > K$. Then $K - 1 < A - 1 = (\tau(b_i^k), \ldots, \tau(b_i^1))_2$. Based on the properties of binary numbers, we know that $(A-1)^j = 1$ and $(K-1)^j = 0$ at the most significant bit j where the values differ. As $\tau(\textsc{DefB}(i,j')) = 1$, we have $\tau(B_i^k) = 0$, a contradiction. □

Lemma 13 *(Condition 2 of Proposition 1). cl_τ respects the infinite values of W for all solutions τ to F^3.*

Proof. If $W(i,j) = \infty$, then τ has to assign $\tau(b_i^n) = \tau(b_j^n)$ for each $n = 1..k$ in order to satisfy the hard clauses corresponding to $b_i^n \leftrightarrow b_j^n$. Hence $\tau(b_i^n) = \tau(b_j^n)$ for all bits and $cl_\tau(v_i) = cl_\tau(v_j)$. If $W(i,j) = -\infty$, then $\tau(EQ_{ij}^n) = 0$ for some $n = 1..k$ due to the hard clause $(\neg EQ_{ij}^1 \vee \ldots \vee \neg EQ_{ij}^k)$. It follows that $\tau(b_i^n \leftrightarrow b_j^n) = 0$. Thus $\tau(b_i^n) \neq \tau(b_j^n)$ and $cl_\tau(v_i) \neq cl_\tau(v_j)$. □

Lemma 14 *(Condition 3 of Proposition 1). Let $cl\colon V \to \{1,2,\ldots,K\}$ be any clustering of V that respects the infinite values of W. There is a solution τ to F^3 such that $cl = cl_\tau$.*

Proof. Construct such τ as

$$\tau(b_i^n) = (cl(v_i)-1)^n$$
$$\tau(EQ_{ij}^n) = \begin{cases} 1 \text{ if } (cl(v_i)-1)^n = (cl(v_j)-1)^n \\ 0 \text{ else} \end{cases}$$
$$\tau(S_{ij}) = \begin{cases} 1 \text{ if } (cl(v_i)-1)^t = (cl(v_j)-1)^t \text{ for all } 1 \le t \le k \\ 0 \text{ else} \end{cases}$$
$$\tau(B_i^1) = \begin{cases} 1 \text{ if } K^1 = 1 \text{ and } (cl(v_i)-1)^1 = 0 \\ 0 \text{ else} \end{cases}$$
$$\tau(B_i^n) = \begin{cases} 1 \text{ if } K^n = 1, (cl(v_i)-1)^n = 0 \text{ or } (cl(v_i)-1)^m = K^m \text{ and } \tau(B_i^{n-1}) = 1 \\ 0 \text{ else .} \end{cases}$$

Clearly $cl_\tau = cl$ as long as τ is a solution to F^3, so it remains to be shown that $\tau(F_h^3) = 1$. Consider the different types of hard constraints present in F^3.

1. For any $v_i \in V$ and any bit n, the fact that $\tau(\textsc{DefB}(i,n)) = 1$ follows directly from the definition given in Equation (6), recalling that $(cl(v_i)-1)^n = \tau(b_i^n)$. Furthermore, $cl(v_i) \le K \Leftrightarrow cl(v_i) - 1 < K$. Hence there is a bit position n for which $K^n = 1$, $(cl(v_i)-1)^n = 0$ and $(cl(v_i)-1)^m = K^m$ for all $n < m \le k$. Thus $\tau(B_i^n) = 1$ and $\tau(\textsc{ClustersLessThan}(i,K)) = 1$.
2. For any $W(i,j) < \infty$, $W(i,j) \neq 0$, and any bit n position, it holds that
$$\tau(EQ_{ij}^n) = 1 \Leftrightarrow (cl(v_i)-1)^n = (cl(v_j)-1)^n \Leftrightarrow \tau(b_i^n) = \tau(b_j^n).$$
Hence $\tau(\textsc{Equality}(i,j,m)) = 1$.
3. For any $W(i,j) \neq 0$ and $W(i,j) \in \mathbb{R}$, it holds that
$$\tau(S_{ij}) = 1 \Leftrightarrow \tau(b_i^t) = (cl(v_i)-1)^t = (cl(v_j)-1)^t = \tau(b_j^t) \quad 1 \le t \le k \Leftrightarrow$$
$$\tau(EQ_{ij}^t) = 1 \quad 1 \le t \le k.$$
Hence $\tau(\textsc{SameCluster}(i,j)) = 1$.
4. For all $W(i,j) = \infty$, $cl(v_i) = cl(v_j)$, and hence $cl(v_i) - 1 = cl(v_j) - 1$. By the construction of τ, we have $\tau(b_i^n) = \tau(b_j^n)$ for all n. Thus $\tau(y_i^n \leftrightarrow y_j^n) = 1$ for all bit positions, and $\tau(\mathrm{ML}^B(v_i, v_j)) = 1$.
5. For all $W(i,j) = -\infty$, $cl(v_i) \neq cl(v_j)$ and $cl(v_i) - 1 \neq cl(v_j) - 1$. Hence there is a bit position n for which $(cl(v_i)-1)^n \neq (cl(v_j)-1)^n$. By the construction of τ, $\tau(EQ_{ij}^n) = 0$ and $\tau(\neg EQ_{ij}^1 \vee \ldots \vee \neg EQ_{ij}^k) = 1$. As we already demonstrated that $\tau(\textsc{Equality}(i,j,m)) = 1$ holds for all m, we conclude that $\tau(\mathrm{CL}^B(v_i, v_j)) = 1$. □

Lemma 15 *(Condition 4 of Proposition 1). $\textsc{cost}(F^3, \tau) = H(W, cl_\tau)$ for any solution τ to F^3.*

Proof. As the semantics of the S_{ij} variables exactly match the x_{ij} variables from the transitive encoding, the proof of this lemma is almost identical to the proof of the corresponding result for the transitive encoding. The key observation is that any pair of points v_i and v_j increases the cost of a MaxSAT solution by $|W(i,j)|$ iff it also increases the cost of cl_τ by $|W(i,j)|$. □

References

[1] J. Berg, M. Järvisalo, Optimal correlation clustering via MaxSAT, in: Proc. 2013 IEEE ICDM Workshops, IEEE Press, 2013, pp. 750–757.
[2] D.H. Fisher, Knowledge acquisition via incremental conceptual clustering, Mach. Learn. 2 (2) (1987) 139–172.
[3] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Comput. Surv. 31 (3) (1999) 264–323.
[4] N. Bansal, A. Blum, S. Chawla, Correlation clustering, Mach. Learn. 56 (1–3) (2004) 89–113.
[5] N. Ailon, M. Charikar, A. Newman, Aggregating inconsistent information: ranking and clustering, J. ACM 55 (5), Article No. 23.
[6] M. Charikar, V. Guruswami, A. Wirth, Clustering with qualitative information, J. Comput. Syst. Sci. 71 (3) (2005) 360–383.
[7] R. Shamir, R. Sharan, D. Tsur, Cluster graph modification problems, Discrete Appl. Math. 144 (1–2) (2004) 173–182.
[8] I. Giotis, V. Guruswami, Correlation clustering with a fixed number of clusters, Theory Comput. 2 (1) (2006) 249–266.
[9] E.D. Demaine, D. Emanuel, A. Fiat, N. Immorlica, Correlation clustering in general weighted graphs, Theor. Comput. Sci. 361 (2–3) (2006) 172–187.
[10] J.V. Gael, X. Zhu, Correlation clustering for crosslingual link detection, in: Proc. IJCAI, 2007, pp. 1744–1749.
[11] A. Ben-Dor, R. Shamir, Z. Yakhini, Clustering gene expression patterns, J. Comput. Biol. 6 (3/4) (1999) 281–297.
[12] F. Bonchi, A. Gionis, A. Ukkonen, Overlapping correlation clustering, in: Proc. ICDM, IEEE, 2011, pp. 51–60.
[13] F. Bonchi, A. Gionis, F. Gullo, A. Ukkonen, Chromatic correlation clustering, in: Proc. KDD, ACM, 2012, pp. 1321–1329.
[14] N. Cesa-Bianchi, C. Gentile, F. Vitale, G. Zappella, A correlation clustering approach to link classification in signed networks, in: Proc. COLT, in: J. Mach. Learn. Res. Workshop Conf. Proc., vol. 23, JMLR.org, 2012, pp. 34.1–34.20.
[15] P. Bonizzoni, G.D. Vedova, R. Dondi, T. Jiang, Correlation clustering and consensus clustering, in: Proc. ISAAC, in: Lecture Notes in Computer Science, vol. 3827, Springer, 2005, pp. 226–235.
[16] V. Filkov, S. Skiena, Integrating microarray data by consensus clustering, Int. J. Artif. Intell. Tools 13 (4) (2004) 863–880.
[17] V. Filkov, S. Skiena, Heterogeneous data integration with the consensus clustering formalism, in: Proc. DILS, in: Lecture Notes in Computer Science, vol. 2994, Springer, 2004, pp. 110–123.
[18] R. Giancarlo, F. Utro, Speeding up the consensus clustering methodology for microarray data analysis, Algorithms Mol. Biol. 6 (2011) 1.
[19] Z. Yu, H.-S. Wong, H.-Q. Wang, Graph-based consensus clustering for class discovery from gene expression data, Bioinformatics 23 (21) (2007) 2888–2896.
[20] T. Guns, S. Nijssen, L.D. Raedt, Itemset mining: a constraint programming perspective, Artif. Intell. 175 (12–13) (2011) 1951–1983.
[21] S. Nijssen, T. Guns, L.D. Raedt, Correlated itemset mining in ROC space: a constraint programming approach, in: Proc. KDD, ACM, 2009, pp. 647–656.
[22] L.D. Raedt, T. Guns, S. Nijssen, Constraint programming for data mining and machine learning, in: Proc. AAAI, AAAI Press, 2010.
[23] I. Davidson, S.S. Ravi, L. Shamis, A SAT-based framework for efficient constrained clustering, in: Proc. SDM, SIAM, 2010, pp. 94–105.
[24] T. Guns, S. Nijssen, L.D. Raedt, K-pattern set mining under constraints, IEEE Trans. Knowl. Data Eng. 25 (2) (2013) 402–418.
[25] B. Négrevergne, A. Dries, T. Guns, S. Nijssen, Dominance programming for itemset mining, in: Proc. ICDM, IEEE, 2013, pp. 557–566.
[26] S. Gilpin, S. Nijssen, I.N. Davidson, Formalizing hierarchical clustering as integer linear programming, in: Proc. AAAI, AAAI Press, 2013.
[27] K. Wagstaff, C. Cardie, Clustering with instance-level constraints, in: Proc. ICML, Morgan Kaufmann, 2000, pp. 1103–1110.
[28] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, Constrained K-means clustering with background knowledge, in: Proc. ICML, Morgan Kaufmann, 2001, pp. 577–584.
[29] I. Davidson, S.S. Ravi, Intractability and clustering with constraints, in: Proc. ICML, ACM, 2007, pp. 201–208.
[30] A. Biere, M.J.H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, IOS Press, 2009.
[31] J.-P. Métivier, P. Boizumault, B. Crémilleux, M. Khiari, S. Loudni, Constrained clustering using SAT, in: Proc. IDA, in: Lecture Notes in Computer Science, vol. 7619, Springer, 2012, pp. 207–218.
[32] C.M. Li, F. Manyà, MaxSAT, hard and soft constraints, in: Handbook of Satisfiability, IOS Press, 2009, pp. 613–631.
[33] P. Bonizzoni, G.D. Vedova, R. Dondi, T. Jiang, On the approximation of correlation clustering and consensus clustering, J. Comput. Syst. Sci. 74 (5) (2008) 671–696.
[34] M. Charikar, A. Wirth, Maximizing quadratic programs: extending Grothendieck's inequality, in: Proc. FOCS, IEEE Computer Society, 2004, pp. 54–60.
[35] D. Klein, S.D. Kamvar, C.D. Manning, From instance-level constraints to space-level constraints: making the most of prior knowledge in data clustering, in: Proc. ICML, Morgan Kaufmann, 2002, pp. 307–314.
[36] I. Davidson, S.S. Ravi, Clustering with constraints: feasibility issues and the k-means algorithm, in: Proc. SDM, SIAM, 2005, pp. 138–149.
[37] I. Davidson, S.S. Ravi, The complexity of non-hierarchical clustering with instance and cluster level constraints, Data Min. Knowl. Discov. 14 (1) (2007) 25–61.
[38] M. Křivánek, J. Morávek, NP-hard problems in hierarchical-tree clustering, Acta Inform. 23 (3) (1986) 311–323.
[39] Y. Chen, S. Safarpour, J. Marques-Silva, A.G. Veneris, Automated design debugging with maximum satisfiability, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29 (11) (2010) 1804–1817.
[40] C.S. Zhu, G. Weissenbacher, S. Malik, Post-silicon fault localisation using maximum satisfiability and backbones, in: Proc. FMCAD, FMCAD Inc., 2011, pp. 63–66.
[41] M. Jose, R. Majumdar, Cause clue clauses: error localization using maximum satisfiability, in: Proc. PLDI, ACM, 2011, pp. 437–446.
[42] J. Guerra, I. Lynce, Reasoning over biological networks using maximum satisfiability, in: Proc. CP, in: Lecture Notes in Computer Science, vol. 7514, Springer, 2012, pp. 941–956.
[43] J. Berg, M. Järvisalo, B. Malone, Learning optimal bounded treewidth Bayesian networks via maximum satisfiability, in: Proc. AISTATS, vol. 33, JMLR, 2014, pp. 86–95.
[44] M. Järvisalo, D. Le Berre, O. Roussel, L. Simon, The international SAT solver competitions, AI Mag. 33 (1) (2012) 89–92.
[45] C.M. Li, F. Manyà, N.O. Mohamedou, J. Planes, Exploiting cycle structures in Max-SAT, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 5584, Springer, 2009, pp. 467–480.
[46] M. Koshimura, T. Zhang, H. Fujita, R. Hasegawa, QMaxSAT: a partial Max-SAT solver, J. Satisf. Boolean Model. Comput. 8 (1/2) (2012) 95–100.
[47] J. Marques-Silva, J. Planes, Algorithms for maximum satisfiability using unsatisfiable cores, in: Proc. DATE, IEEE, 2008, pp. 408–413.
[48] A. Morgado, F. Heras, J. Marques-Silva, Improvements to core-guided binary search for MaxSAT, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 7317, Springer, 2012, pp. 284–297.
[49] F. Heras, A. Morgado, J. Marques-Silva, Core-guided binary search algorithms for maximum satisfiability, in: Proc. AAAI, AAAI Press, 2011.
[50] C. Ansótegui, M.L. Bonet, J. Levy, SAT-based MaxSAT algorithms, Artif. Intell. 196 (2013) 77–105.
[51] A. Morgado, F. Heras, M.H. Liffiton, J. Planes, J. Marques-Silva, Iterative and core-guided MaxSAT solving: a survey and assessment, Constraints 18 (4) (2013) 478–534.
[52] Z. Fu, S. Malik, On solving the partial MaxSAT problem, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 4121, Springer, 2006, pp. 252–265.
[53] V.M. Manquinho, J.P.M. Silva, J. Planes, Algorithms for weighted boolean optimization, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 5584, Springer, 2009, pp. 495–508.
[54] A. Morgado, C. Dodaro, J. Marques-Silva, Core-guided MaxSAT with soft cardinality constraints, in: Proc. CP, in: Lecture Notes in Computer Science, vol. 8656, Springer, 2014, pp. 564–573.

[55] J. Davies, F. Bacchus, Exploiting the power of MIPs solvers in MaxSAT, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 7962, Springer, 2013, pp. 166–181.
[56] C. Ansótegui, J. Gabàs, Solving (weighted) partial MaxSAT with ILP, in: Proc. CPAIOR, in: Lecture Notes in Computer Science, vol. 7874, Springer, 2013, pp. 403–409.
[57] J.P. Marques-Silva, I. Lynce, Towards robust CNF encodings of cardinality constraints, in: Proc. CP, in: Lecture Notes in Computer Science, vol. 4741, Springer, 2007, pp. 483–497.
[58] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, A parametric approach for smaller and better encodings of cardinality constraints, in: Proc. CP, vol. 8124, Springer, 2013, pp. 80–96.
[59] S. Prestwich, CNF encodings, in: Handbook of Satisfiability, IOS Press, 2009, pp. 75–97, Ch. 2.
[60] C. Sinz, Towards an optimal CNF encoding of boolean cardinality constraints, in: Proc. CP, in: Lecture Notes in Computer Science, vol. 3709, 2005, pp. 827–831.
[61] F. Heras, A. Morgado, J. Marques-Silva, An empirical study of encodings for group MaxSAT, in: Proc. Canadian Conference on AI, in: Lecture Notes in Computer Science, vol. 7310, Springer, 2012, pp. 85–96.
[62] T. Nepusz, R. Sasidharan, A. Paccanaro, SCPS: a fast implementation of a spectral method for detecting protein families on a genome-wide scale, BMC Bioinform. 11 (2010) 120.
[63] S. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman, Basic local alignment search tool, J. Mol. Biol. 215 (3) (1990) 403–410.
[64] J. Davies, F. Bacchus, Solving MaxSAT by solving a sequence of simpler SAT instances, in: Proc. CP, in: Lecture Notes in Computer Science, vol. 6876, Springer, 2011, pp. 225–239.
[65] J. Davies, F. Bacchus, Postponing optimization to speed up MAXSAT solving, in: Proc. CP, in: Lecture Notes in Computer Science, vol. 8124, Springer, 2013, pp. 247–262.
[66] T. Achterberg, T. Berthold, T. Koch, K. Wolter, Constraint integer programming: a new approach to integrate CP and MIP, in: Proc. CPAIOR, in: Lecture Notes in Computer Science, vol. 5015, Springer, 2008, pp. 6–20.
[67] A. Wirth, Correlation clustering, in: Encyclopedia of Machine Learning, Springer, 2010, pp. 227–231.
[68] J. Sturm, Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones, Optim. Methods Softw. 11–12 (1999) 625–653, version 1.05 available from http://fewcal.kub.nl/sturm.
[69] C. Buchheim, M.D. Santis, L. Palagi, A fast branch-and-bound algorithm for non-convex quadratic integer optimization subject to linear constraints using ellipsoidal relaxations, Oper. Res. Lett. 43 (4) (2015) 384–388.
[70] D. Cohn, R. Caruana, A. McCallum, Semi-supervised clustering with user feedback, Tech. rep., 2003.
[71] I. Davidson, S.S. Ravi, M. Ester, Efficient incremental constrained clustering, in: P. Berkhin, R. Caruana, X. Wu (Eds.), KDD, ACM, 2007, pp. 240–249.
[72] W. Rand, Objective criteria for the evaluation of clustering methods, J. Am. Stat. Assoc. 66 (336) (1971) 846–850.
[73] A. Belov, A. Morgado, J. Marques-Silva, SAT-based preprocessing for MaxSAT, in: Proc. LPAR, in: Lecture Notes in Computer Science, vol. 8312, Springer, 2013, pp. 96–111.
[74] J. Berg, P. Saikko, M. Järvisalo, Improving the effectiveness of SAT-based preprocessing for MaxSAT, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015, AAAI Press, 2015.
[75] A. Belov, M. Järvisalo, J. Marques-Silva, Formula preprocessing in MUS extraction, in: Proc. TACAS, in: Lecture Notes in Computer Science, vol. 7795, Springer, 2013, pp. 108–123.
[76] N. Manthey, Coprocessor 2.0 – a flexible CNF simplifier (tool presentation), in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 7317, Springer, 2012, pp. 436–441.
[77] N. Eén, A. Biere, Effective preprocessing in SAT through variable and clause elimination, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 3569, Springer, 2005, pp. 61–75.
[78] M. Järvisalo, A. Biere, M. Heule, Blocked clause elimination, in: Proc. TACAS, in: Lecture Notes in Computer Science, vol. 6015, Springer, 2010, pp. 129–144.
[79] M. Heule, M. Järvisalo, A. Biere, Clause elimination procedures for CNF formulas, in: Proc. LPAR, in: Lecture Notes in Computer Science, vol. 6397, Springer, 2010, pp. 357–371.
[80] M. Heule, M. Järvisalo, A. Biere, Efficient CNF simplification based on binary implication graphs, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 6695, 2011, pp. 201–215.
[81] M. Järvisalo, A. Biere, Reconstructing solutions after blocked clause elimination, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 6175, Springer, 2010, pp. 340–345.
[82] N. Narodytska, F. Bacchus, Maximum satisfiability using core-guided MaxSAT resolution, in: Proc. AAAI, AAAI Press, 2014, pp. 2717–2723.
[83] R. Martins, V.M. Manquinho, I. Lynce, Open-WBO: a modular MaxSAT solver, in: Proc. SAT, in: Lecture Notes in Computer Science, vol. 8561, Springer, 2014, pp. 438–445.
[84] R. Martins, S. Joshi, V.M. Manquinho, I. Lynce, Incremental cardinality constraints for MaxSAT, in: Proc. CP, in: Lecture Notes in Computer Science, vol. 8656, Springer, 2014, pp. 531–548.
[85] J. Larrosa, F. Heras, Resolution in Max-SAT and its relation to local consistency in weighted CSPs, in: Proc. IJCAI, Professional Book Center, 2005, pp. 193–198.
[86] E.D. Demaine, N. Immorlica, Correlation clustering with partial information, in: Proc. RANDOM-APPROX, in: Lecture Notes in Computer Science, vol. 2764, Springer, 2003, pp. 1–13.
[87] M.X. Goemans, D.P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, J. ACM 42 (6) (1995) 1115–1145.
[88] A. McCallum, B. Wellner, Conditional models of identity uncertainty with application to noun coreference, in: Proc. NIPS, 2004, pp. 905–912.
[89] L. Drummond, R. Figueiredo, Y. Frota, M. Levorato, Efficient solution of the correlation clustering problem: an application to structural balance, in: Proc. OTM Workshops, in: Lecture Notes in Computer Science, vol. 8186, Springer, 2013, pp. 674–683.
[90] B. Babaki, T. Guns, S. Nijssen, Constrained clustering using column generation, in: Proc. CPAIOR, in: Lecture Notes in Computer Science, vol. 8451, Springer, 2014, pp. 438–454.
[91] A. Bar-Hillel, T. Hertz, N. Shental, D. Weinshall, Learning distance functions using equivalence relations, in: Proc. ICML, AAAI Press, 2003, pp. 11–18.
[92] I. Davidson, S.S. Ravi, Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results, Data Min. Knowl. Discov. 18 (2) (2009) 257–282.
[93] T. Coleman, J. Saunderson, A. Wirth, Spectral clustering with inconsistent advice, in: Proc. ICML, ACM, New York, NY, USA, 2008, pp. 152–159.
[94] M. Mueller, S. Kramer, Integer linear programming models for constrained clustering, in: Proc. DS, in: Lecture Notes in Computer Science, vol. 6332, Springer, 2010, pp. 159–173.
[95] J. Schmidt, E.M. Brändle, S. Kramer, Clustering with attribute-level constraints, in: Proc. ICDM, IEEE, 2011, pp. 1206–1211.
[96] D. Aloise, P. Hansen, L. Liberti, An improved column generation algorithm for minimum sum-of-squares clustering, Math. Program. 131 (1–2) (2012) 195–220.
[97] T.-B.-H. Dao, K.-C. Duong, C. Vrain, A declarative framework for constrained clustering, in: Proc. ECML-PKDD, 2013, pp. 419–434.

Paper VI

Jeremias Berg, Matti Järvisalo, and Brandon Malone

Learning Optimal Bounded Treewidth Bayesian Networks via Maximum Satisfiability

Learning Optimal Bounded Treewidth Bayesian Networks via Maximum Satisfiability

Jeremias Berg and **Matti Järvisalo** and **Brandon Malone**
HIIT & Department of Computer Science, University of Helsinki, Finland

Abstract

Bayesian network structure learning is the well-known computationally hard problem of finding a directed acyclic graph structure that optimally describes given data. A learned structure can then be used for probabilistic inference. While exact inference in Bayesian networks is in general NP-hard, it is tractable in networks with low treewidth. This provides good motivations for developing algorithms for the NP-hard problem of learning optimal bounded treewidth Bayesian networks (BTW-BNSL). In this work, we develop a novel score-based approach to BTW-BNSL, based on casting BTW-BNSL as weighted partial Maximum satisfiability. We demonstrate empirically that the approach scales notably better than a recent exact dynamic programming algorithm for BTW-BNSL.

1 INTRODUCTION

Bayesian networks are an important and widely-used class of probabilistic graphical models for representing joint probability distributions, i.e., probabilistic relationships among a set of variables of interest (Pearl, 1988). A Bayesian network consists of a network structure, represented as an acyclic directed graph (DAG), and the parameters associated with each node (i.e., variable) in the DAG. Most often, a Bayesian network that represents given data well is not known *a priori*, and hence needs to be learned from data. Given a network structure and complete data, determining the parameters of the variables is simple, whereas learning the DAG structure, i.e., the Bayesian network structure learning problem (BNSL), is computationally challenging.

In this work we focus on the BNSL problem within the widely studied score-based framework, in which a score is assigned to each DAG structure, and the goal is to find a best-scoring network. The structure learning problem is NP-complete in general (Chickering, 1996), which justified the fact that most early work on BNSL focused on local search algorithms, such as greedy hill climbing in the space of DAGs (Heckerman, 1998), equivalence classes of DAGs (Chickering, 2002), or over variable orderings (Teyssier and Koller, 2005), and local searching over constraint optimization formulations of BNSL (Cussens, 2008).

After learning a Bayesian network, the network is typically used for probabilistic inference tasks, such as determining the most likely joint assignments of a set of variables under given evidence. In order to accurately answer such queries, it is important to learn a network that explains the input data well. Throughout the last decade, there has been increasing interest in developing algorithms for *optimally* solving BNSL, and a variety of algorithms which are guaranteed to find a network structure with optimal score have been proposed (Ott and Miyano, 2003; Koivisto and Sood, 2004; Silander and Myllymäki, 2006; Cussens, 2011; Yuan and Malone, 2013).

While exact Bayesian inference is in general NP-hard (Cooper, 1990), for *bounded (fixed) treewidth* networks exact inference becomes tractable (Lauritzen and Spiegelhalter, 1988). This motivates the study of algorithms for the problem of learning optimal bounded treewidth Bayesian networks (BTW-BNSL). Despite the recent progress in practical algorithms for optimally solving BNSL without treewidth constraints, very few practical algorithms have been proposed for learning network structures under restrictions on the treewidth of the networks (Elidan and Gould, 2008; Korhonen and Parviainen, 2013); the only approach learning optimal bounded treewidth network structures is the recent exact dynamic programming algorithm of Korhonen and Parviainen

Appearing in Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33. Copyright 2014 by the authors.

(2013).

Much like the general BNSL problem, BTW-BNSL is NP-hard (Korhonen and Parviainen, 2013): more precisely, BTW-BNSL(W), the problem of finding an optimal Bayesian network structure of treewidth at most W, is NP-hard for any *fixed* $W \geq 2$ (DAGs with $W = 1$ being trees). Indeed, the restriction on the treewidth of the DAG structures is a *non-trivial* additional constraint over the general BNSL problem, which poses challenges for developing algorithms for BTW-BNSL.

In this work, we develop a novel score-based approach to learning optimal bounded treewidth Bayesian network structures. Our approach is based on casting BTW-BNSL for a given bound W on the treewidth of the DAG structures of interest as an abstract combinatorial optimization problem. More precisely, we present an intricate encoding of BTW-BNSL as weighted partial Maximum Satisfiability (MaxSAT in short). The encoding ensures that the optimal solutions of the MaxSAT instance encoding an arbitrary instance of BTW-BNSL(W) correspond to optimal DAG structures wrt a given scoring function. For finding optimal structures using the MaxSAT encoding, we employ a state-of-the-art MaxSAT solver extended to real-valued costs for exactly encoding the local scores. We demonstrate empirically that our approach scales notably better than the recent exact dynamic programming algorithm for BTW-BNSL (Korhonen and Parviainen, 2013) on standard BNSL benchmarks and for different values of W. Furthermore, in view of practical efficiency, our approach can benefit from foreseeable future improvements in state-of-the-art MaxSAT solver technology. The approach is applicable under any decomposable scoring function (Heckerman, 1998), i.e., scoring functions in which the score for an entire network is the sum of the local scores for the chosen parent sets for the individual variables in the network, including e.g. the commonly used scoring functions MDL (Lam and Bacchus, 1994), BD (Cooper and Herskovits, 1992; Heckerman et al., 1995), and fNML (Silander et al., 2008).

2 PRELIMINARIES

In order to formally define the problem of learning optimal bounded treewidth Bayesian network structures, we first define necessary concepts related to treewidth and tree-decompositions. We also give necessary background on MaxSAT.

2.1 Treewidth

The treewidth of an undirected graph G is defined in terms of the *tree-decompositions* of G.

Definition 1 *A tree-decomposition of an undirected graph $G = (V, E)$ is a tree T over a set $\{V_1, \dots, V_m\}$ of nodes, where $V_i \subseteq V$, with the following properties.*

1. *$\cup_{i=1}^{m} V_i = V$.*
2. *If $\{u, v\} \in E$, then $u, v \in V_i$ for some $i \in \{1, \dots, m\}$.*
3. *For all $i, j, k \in \{1, \dots, m\}$, the following holds: if V_j is on the (unique) path from V_i to V_k in T, then $V_i \cap V_k \subseteq V_j$.*

The width of a tree-decomposition is $\max_{i=1}^{m} |V_i| - 1$.

Definition 2 *The treewidth $tw(G)$ of an undirected graph $G = (V, E)$ is the minimum width over all tree-decompositions of G.*

It is well-known that, for any undirected graph $G = (V, E)$, any linear ordering of the nodes V of G defines a tree-decomposition of G, and that there is always an "optimal" linear ordering of V defining an *optimal* tree-decomposition, i.e., a tree-decomposition of width $tw(G)$ (Dechter, 1999; Bodlaender, 2005). Furthermore, without needing to explicitly construct the corresponding optimal tree-decomposition, the treewidth of G can be determined based on an optimal linear ordering $\prec$ of V. A node $v_i \in V$ is a *predecessor* of $v_j \in V$ under $\prec$ if $i \prec j$ and $\{v_i, v_j\} \in E$; v_i is a *successor* of v_j under $\prec$ if $j \prec i$ and $\{v_i, v_j\} \in E$. Given a linear ordering $\prec$ of V, the width of the corresponding tree-decomposition is determined by applying the following *triangulation* procedure on G under $\prec$: For each pair v_i, v_j of nodes in V, add the edge $\{v_i, v_j\}$ to E if v_i and v_j have a common predecessor. Repeat this as long as new edges can be added to E. We denote the resulting edge-relation by $\Delta(E, \prec)$, defining the *triangulation* $\Delta(G, \prec) = (V, \Delta(E, \prec))$ of G under $\prec$. Orienting the edges of $\Delta(G, \prec)$ according to $\prec$ gives the directed edge-relation

$$\vec{\Delta}(E, \prec) = \{(v_i, v_j) \mid \{v_i, v_j\} \in \Delta(E, \prec),\ i \prec j\}$$

defining the *ordered graph* $\vec{\Delta}(G, \prec) = (V, \vec{\Delta}(E, \prec))$ of G under $\prec$. Now, the width of the tree-decomposition defined by $\prec$ is

$$\max_{v_i \in V} |\{(v_i, v_j) \in \vec{\Delta}(E, \prec)\}|, \tag{1}$$

i.e., the maximum number of successors over all nodes in $\Delta(E, \prec)$. The treewidth $tw(G)$ of G is then

$$\min_{\prec} \max_{v_i \in V} |\{(v_i, v_j) \in \vec{\Delta}(E, \prec)\}|, \tag{2}$$

over all linear orderings $\prec$ of the nodes V of G.

Before a concrete example of triangulation and ordered graphs, we proceed by defining the treewidth for the DAG structures of Bayesian networks.

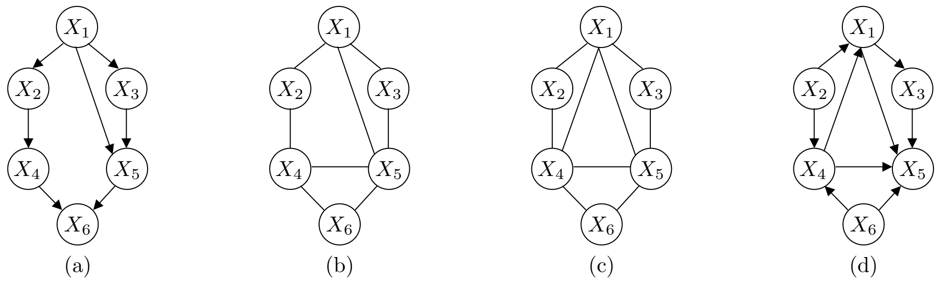

Figure 1: Example: (a) a DAG $G = (X = \{X_1, \ldots, X_6\}, E)$; (b) the moralized graph $\textsc{Moral}(G) = (X, \mathrm{M}(E))$ of G; (c) the triangulation $\Delta(\textsc{Moral}(G), \prec)$ of the moralized graph under the linear ordering $X_6 \prec X_2 \prec X_4 \prec X_1 \prec X_3 \prec X_5$; (d) the ordered graph $\vec{\Delta}(\textsc{Moral}(G), \prec)$.

2.2 Bounded Treewidth Bayesian Network Structure Learning

Given a set $X = \{X_1, \ldots, X_N\}$ of nodes (representing random variables), an element of $\mathcal{P}_i = 2^{X \setminus \{X_i\}}$ is a *candidate parent set* of X_i. For a given DAG $G = (X, E)$, the parent set of node X_i is $\{X_j \mid (X_j, X_i) \in E\}$, i.e., consists of the parents of X_i in G. Picking a single $P_i \in \mathcal{P}_i$ for each X_i gives rise to the (not necessarily acyclic) graph in which, for each X_i, there is an edge (X_j, X_i) iff $X_j \in P_i$.

The treewidth of a Bayesian network structure is defined as the treewidth of the *moralized graph* induced by the DAG structure of the network. This is motivated by the fact that Bayesian inference is tractable in structures whose moralized graph has bounded treewidth, forming the basis for exact join-tree inference algorithms (Lauritzen and Spiegelhalter, 1988).

Definition 3 *Given a DAG $G = (X, E)$, the* moralized graph $\textsc{Moral}(G) = (X, \mathrm{M}(E))$ *induced by G is an undirected graph defined by the edge relation*

$$\begin{aligned}\mathrm{M}(E) = &\{\{X_i, X_j\} \mid (X_i, X_j) \in E\} \cup \\ &\{\{X_i, X_j\} \mid \exists k \ s.t.\ (X_i, X_k), (X_j, X_k) \in E\}.\end{aligned}$$

In words, the moralized graph contains an undirected version of each edge in the DAG, and an edge between every pair of nodes which have a common child in the DAG.

The treewidth of the DAG structure G of any Bayesian network can be determined by finding a linear ordering $\prec$ that minimizes Eq. 1 for the ordered graph $\vec{\Delta}(\textsc{Moral}(G), \prec)$ of the moralization $\textsc{Moral}(G)$ of G under $\prec$. We denote by $\textsc{tw}(W)$ the class of DAGs having treewidth at most W.

As an example, Figure 1 illustrates for (a) a given DAG $G = (X, E)$ (b) the moralized graph $\textsc{Moral}(G)$, and, for a given linear ordering $\prec$ of the nodes X, (c) the triangulation $\Delta(\textsc{Moral}(G), \prec)$ and (d) the ordered graph $\vec{\Delta}(\textsc{Moral}(G), \prec)$. For this ordering $\prec$, Eq. 1 evaluates to 2, and hence $G \in \textsc{tw}(2)$. In fact, it can be checked that this $\prec$ defines an optimal tree-decomposition of G, minimizing Eq. 2, which implies that $tw(G) = 2$.

With these definitions, we can formally state the bounded treewidth Bayesian network structure learning problem (BTW-BNSL) as follows[1].

The BTW-BNSL Problem

Input: A set $X = \{X_1, \ldots, X_N\}$ of nodes, an integer W, and for each X_i a non-negative local score (cost) $s_i(P_i)$ for each $P_i \in \mathcal{P}_i$.

Task: Find a DAG G^* such that

$$G^* \in \operatorname*{arg\,min}_{G \in \textsc{tw}(W)} \sum_{i=1}^{N} s_i(P_i), \tag{3}$$

where P_i is the parent set of X_i in G.

Note that the $\mathcal{P}_i$s can be assumed to contain only parent sets P_i with $|P_i| \leq W$, since the treewidth of any DAG containing a node having more than W parents is greater than W. However, the opposite does not hold, i.e., the treewidth of a DAG with at most W parents for each node can still be greater than W.

2.3 Maximum Satisfiability

We shortly review necessary background on Maximum satisfiability (Li and Manyà, 2009).

For a Boolean variable x, there are two literals, x and

[1]The problem can equivalently be defined as a maximization problem under non-positive local scores.

$\neg x$. A clause is a disjunction ($\vee$, logical OR) of literals. A truth assignment is a function from Boolean variables to $\{0, 1\}$. A clause C is satisfied by a truth assignment τ ($\tau(C) = 1$) if $\tau(x) = 1$ for a literal x in C, or $\tau(x) = 0$ for a literal $\neg x$ in C. A set F of clauses is satisfiable if there is an assignment τ satisfying all clauses in F ($\tau(F) = 1$), and unsatisfiable ($\tau(F) = 0$ for any assignment τ) otherwise. An instance $F = (F_h, F_s, c)$ of the *weighted partial MaxSAT* problem consists of two sets of clauses, a set F_h of *hard* clauses and a set F_s of *soft* clauses, and a function $c : F_s \to \mathbb{R}^+$ that associates a non-negative cost with each of the soft clauses.[2] Any truth assignment τ that satisfies F_h is a *solution* to F. The *cost* of a solution τ to F is

$$\textsc{cost}(F, \tau) = \sum_{\substack{C \in F_s: \\ \tau(C)=0}} c(C),$$

i.e., as the sum of the costs of the soft clauses not satisfied by τ. A solution τ is (globally) *optimal* for F if $\textsc{cost}(F, \tau) \leq \textsc{cost}(F, \tau')$ holds for any solution τ' to F. The cost of the optimal solutions of F is denoted by $\textsc{opt}(F)$. Given a weighted partial MaxSAT instance F, the weighted partial MaxSAT problem asks to find an optimal solution to F. From here on, we refer to weighted partial MaxSAT instances simply as MaxSAT instances.

Due to recent advances in MaxSAT solvers, i.e., algorithms for (optimally) solving MaxSAT, MaxSAT is a viable approach to finding globally optimal solutions to various optimization problems. In general, the MaxSAT-based approach has two steps. First, a MaxSAT encoding of the problem is developed. For any instance I of the problem, the encoding produces a MaxSAT instance F_I such that any optimal solution to F_I can be mapped to an optimal solution of I. Then, an off-the-shelf MaxSAT solver is used to find an optimal solution to the MaxSAT instance. As SAT solvers continue improving, larger and larger problems can be solved in practice (Järvisalo et al., 2012).

3 BTW-BNSL as MaxSAT

We will now describe an encoding of BTW-BNSL as (weighted partial) MaxSAT.

For the following, we assume an arbitrary input instance of BTW-BNSL, consisting of a set $X = \{X_1, \ldots, X_N\}$ of nodes, a treewidth bound W, and for each X_i a non-negative local score (cost) $s_i(P_i)$ for each $P_i \in \mathcal{P}_i$ with $|P_i| \leq W$. Given $(X, W, \{s_i\}_{i=1}^N)$, our encoding will produce a weighted partial MaxSAT instance $F(X, W, \{s_i\}_{i=1}^N) = (F_h, F_s, c)$ such that any optimal solution to F corresponds to a DAG G^* that is an optimal solution the BTW-BNSL instance $(X, W, \{s_i\}_{i=1}^N)$, and vice versa.

[2] Our definition for the function c is more general than the more standard $c : F_s \to \mathbb{N}^+$, which restricts the costs of soft clauses to be integral. However, in this work we employ a recent MaxSAT solver which allows for assigning real-valued costs to soft clauses.

3.1 Overview

In order to exactly represent the BTW-BNSL instance as a weighted partial MaxSAT instance, we will encode the following constraints:

1. For each X_i, exactly one parent set $P_i \in \mathcal{P}_i$ is chosen.
2. The graph G^*, corresponding to the choice of a parent set P_i for each i, is acyclic.
3. The moralized graph $\textsc{Moral}(G^*)$ of G^* has treewidth $tw(\textsc{Moral}(G^*)) \leq W$.
4. G^* is an optimal solution of the BTW-BNSL instance, i.e., $G^* \in \arg\min_{G \in \textsc{tw}(W)} \sum_{i=1}^N s_i(P_i)$.

Constraints 1 and 2 together enforce that any choice of a single parent set P_i for each variable X_i corresponds to a DAG G^*. Constraint 3 is the most intricate one, and enforces that G^* has treewidth at most W. Constraint 4 represents the objective function (Eq. 3) of BTW-BNSL.

The main variables used in the encoding are summarized in Table 1.

- The variables P_i^S represent for each node X_i the chosen parent set $S \in \mathcal{P}_i$.
- The variables M_{ij} represent the edges in the moralized graph $\textsc{Moral}(G^*)$ of G^*.
- The variables ord_{ij} represent a linear ordering *ord* of the nodes of G^*.
- The variables O_{ij} represent the successors X_j of node X_i in the ordered graph of $\textsc{Moral}(G^*)$ under *ord*.

3.2 Details

We will now detail the MaxSAT encoding of Constraints 1–4, i.e., our MaxSAT encoding of BTW-BNSL. For clarity, we present the various parts of the encoding using propositional logic, instead of directly presenting the corresponding individual clauses.

1: Enforcing Exactly One Parent Set. For each node X_i, exactly one parent set from $\mathcal{P}_i$ must be chosen. This is enforced by introducing for each node X_i the cardinality constraint

$$\sum_{S \in \mathcal{P}_i} P_i^S = 1. \tag{4}$$

Table 1: The main variables used in the MaxSAT encoding of the BTW-BNSL problem.

Boolean variables	**Interpretation**	**Indices**
P_i^S	represent the parent set of each node in G^*: $P_i^S = 1$ iff S is the parent set of node X_i in G^*	for all $i = 1..N$ and $S \in \mathcal{P}_i$
M_{ij}	represent the moralized graph of G^*: $M_{ij} = 1$ iff $\textsc{Moral}(G^*)$ contains the edge $\{X_i, X_j\}$	for all $i, j = 1..N$ such that $i < j$
ord_{ij}	represent a linear ordering ord of the nodes of G^*: $ord_{ij} = 1$ iff node X_i is a predecessor of node X_j in the linear ordering	for all $i, j = 1..N$ such that $i < j$
O_{ij}	represent the ordered graph $\vec{\Delta}(\textsc{Moral}(G^*), ord)$: $O_{ij} = 1$ iff the ordered graph of $\textsc{Moral}(G^*)$ under ord contains the edge (X_i, X_j)	for all $i, j = 1..N$ such that $i \neq j$

Many different ways of representing such special types of cardinality constraints, often called *exactly-one* constraints, as (hard) clauses have been proposed in the literature. Here we use the so-called *improved sequential counter encoding* for representing Eq.(4) as a set of hard clauses; for details on the improved sequential counted encoding, see (Samer and Veith, 2009).

2: Enforcing Acyclicity. For ruling out cyclic graphs, i.e., for ensuring that any solution to the MaxSAT encoding corresponds to a DAG, we apply the idea of associating a unique, pair-wise different *level number* from $\{1, \ldots, N\}$ with each node X_i, and enforce that, given that a parent set $S \in \mathcal{P}_i$ is chosen for X_i, the level number of X_i is greater than the level number of each $X_j \in S$.[3]

We use a binary encoding of the level numbers of the nodes. For each node X_i, $\log_2 N$ Boolean variables $b_i^1, \ldots, b_i^{\log_2 N}$ form the binary representation $b_i^{\log_2 N} \ldots b_i^1$ of the level number of X_i. For a compact encoding, we also use auxiliary variables EQ_{ij}^k and GT_{ij}^k, with the interpretations that $EQ_{ij}^k = 1$ iff $b_i^k = b_j^k$, and $GT_{ij}^k = 1$ iff $b_k^i = 1$, $b_k^j = 0$, and $EQ_{ij}^{k'} = 1$ for all $k' > k$ (i.e., the kth bit is the most significant bit in which the level numbers of X_i and X_j differ, and the level number of X_i is greater than that of X_j). Using these variables, the unique level numbers for the nodes are enforced as follows.

The fact that each node gets a different level number from $\{1, \ldots, N\}$ is enforced by stating that for each pair of distinct nodes X_i, X_j, the level number of X_i is different from that of X_j. This is enforced by

$$\bigvee_{k=1}^{\log_2 N} \neg EQ_{ij}^k, \tag{5}$$

i.e., there is a bit-position k in which the binary representations of the level numbers of X_i and X_j differ.

[3]Variations of the same idea have been applied for enforcing acyclicity with linear integer constraints in different contexts, under e.g. the terms *level rankings* (Niemelä, 2008) and *generation numbers* (Cussens et al., 2013).

Furthermore, if parent set $S \in \mathcal{P}_i \setminus \{\emptyset\}$ is chosen for node X_i, then for each $X_j \in S$, there is a bit position k which is the most significant bit in which the level numbers of X_i and X_j differ, and the level number of X_i is greater than that of X_j:

$$P_i^S \to \bigvee_{k=1}^{\log_2 N} GT_{ij}^k \quad \text{for all } j \text{ s.t. } X_j \in S. \tag{6}$$

The semantics of the variables GT_{ij}^k and EQ_{ij}^k are encoded as

$$GT_{ij}^k \quad \leftrightarrow \quad b_i^k \wedge \neg b_j^k \wedge \bigwedge_{k'=k+1}^{\log_2 N} EQ_{ij}^{k'}, \tag{7}$$

$$EQ_{ij}^k \quad \leftrightarrow \quad \left(b_i^k \leftrightarrow b_j^k\right). \tag{8}$$

While Eqs. 5–8 together with Eq. 4 ensure that any solution corresponds to a DAG, we also include a single additional *redundant* clause, stating the fact that a DAG has at least one root node, i.e., a node X_i with the empty parent set $\emptyset$:

$$\bigvee_{i=1}^{N} P_i^{\emptyset}. \tag{9}$$

While this clause is redundant in that it does not change the set of solutions, it turned out that in practice adding this clause speeds up MaxSAT solving.

3: Enforcing the Treewidth Bound. The most intricate part of the MaxSAT encoding deals with mapping parent sets to the moralized graph of a DAG G^* corresponding to the parent sets, and then enforcing that the moralized graph $\textsc{Moral}(G^*)$ of G^* has treewidth $tw(\textsc{Moral}(G^*)) \leq W$.

(i) From Parent Sets to the Moralized Graph. We directly connect the choices of parent sets, represented by the P_i^S variables, with the edges in the corresponding moralized graph, represented by the variables M_{ij}. The encoding follows closely the definition of moralized graphs (Def. 3). Eq. 10 enforces that, if a particular parent set $S \in \mathcal{P}_i$ is chosen, then in the moralized graph there is (i) an edge between X_i and each

$X_j \in S$, and (ii) an edge between each pair of distinct nodes $X_j, X_k \in S$.

$$P_i^S \rightarrow \bigwedge_{X_j \in S} M_{ij} \wedge \bigwedge_{X_j, X_k \in S} M_{jk}. \tag{10}$$

The opposite direction is encoded as Eq. 11: if there is an edge in the moralized graph between nodes X_i and X_j, it must hold that: (i) X_j is in the parent set of X_i, (ii) X_i is in the parent set of X_j, or (iii) both X_i and X_j are in the parent set of some $X_k \in X \setminus \{X_i, X_j\}$.

$$M_{ij} \rightarrow \bigvee_{S : X_j \in S} P_i^S \vee \bigvee_{S : X_i \in S} P_j^S \vee \bigvee_{\substack{X_k \in X \setminus \{X_i, X_j\} \\ S : X_i, X_j \in S}} P_k^S \tag{11}$$

Notice that, with this encoding, we do not need to introduce explicit Boolean variables for explicitly representing the actual edges of the DAG corresponding to the choice of parent sets.

(ii) Encoding Linear Orderings. For enforcing the treewidth bound on the moralized graphs, we follow—with minor modifications—a SAT encoding of treewidth in undirected graphs presented in (Samer and Veith, 2009). Following Samer and Veith (2009), we do not encode the construction of a tree-decomposition of $\textsc{Moral}(G^*)$ explicitly. Instead, our encoding enforces the condition that for any G^*, there needs to be a linear ordering *ord* of X under which the maximum number of successors over all nodes in the ordered graph of $\textsc{Moral}(G^*)$ is at most W.

The choice of a linear ordering of X is represented by the ord_{ij} variables. For notational convenience, let

$$ord^*_{ij} = \begin{cases} ord_{ij} & \text{if } i < j \\ \neg ord_{ji} & \text{else} \end{cases}.$$

Transitivity of linear orderings is enforced in the encoding by stating for all distinct $i, j, k = 1..N$

$$ord^*_{ij} \wedge ord^*_{jk} \rightarrow ord^*_{ik}. \tag{12}$$

(iii) Bounding Treewidth via Triangulation. Recall that the treewidth of the tree-decomposition corresponding to a linear ordering $\prec$ is $\max_{v_i \in V} |\{\{v_i, v_j\} \in E : i \prec j\}|$, where E is the edge-relation of the triangulated moralized graph; and that the variable O_{ij} represents the fact that the ordered graph of $\textsc{Moral}(G^*)$ under the linear ordering $\prec$ (represented by the ord_{ij} variables) contains the edge (X_i, X_j). It follows that enforcing the cardinality constraint

$$\sum_{j \neq i} O_{ij} \leq W \tag{13}$$

for each $i = 1..N$ is equivalent to the requirement $\max_{v_i \in V} |\{\{v_i, v_j\} \in E \mid i \prec j\}| \leq W$. Again, different ways of representing such general cardinality constraints as clauses have been proposed in the literature. Since here the interesting cases are when W takes values greater than one, we use a compact encoding based on so-called *cardinality networks* (Asín et al., 2011; Abío et al., 2013) for representing the constraints as hard clauses.

What remains is the definition of the O_{ij} variables, i.e., encoding of the ordered graph induced by a linear ordering.

–If the moralized graph contains an edge $\{X_i, X_j\}$, then the triangulation of the moralized graph also contains the edge $\{X_i, X_j\}$, and hence the ordered graph contains either the edge (X_i, X_j) or the edge (X_j, X_i). This is enforced by

$$M_{ij} \rightarrow (O_{ij} \vee O_{ji}) \quad \text{for all } i < j. \tag{14}$$

–If nodes X_i and X_j have a common predecessor in the moralized graph, then the triangulation of the moralized graph contains the edge $\{X_i, X_j\}$, and hence the ordered graph contains either the edge (X_i, X_j) or the edge (X_j, X_i). This is enforced for all distinct $i, j, k = 1..N$ by

$$(O_{ki} \wedge O_{kj}) \rightarrow (O_{ij} \vee O_{ji}). \tag{15}$$

Finally, in both Eqs. 14 and 15, the choice of which of the edges (X_i, X_j) or (X_j, X_i) occur in the ordered graph depends on the linear ordering *ord*. Essentially, O_{ij} must be consistent with ord_{ij} in that, if i comes before j in *ord*, then the edge (X_j, X_i) does not occur in the ordered graph under *ord*:

$$ord^*_{ij} \rightarrow \neg O_{ji}. \tag{16}$$

4: Encoding the Objective Function. We encode the BTW-BNSL objective function (Eq. 3) using soft clauses. Accordingly, choosing a specific parent set $S \in \mathcal{P}_i$ for node X_i should incur a cost equal to the local score $s_i(S)$. Thus, we introduce for each X_i and each $S \in \mathcal{P}_i$ the soft clause

$$(\neg P_i^S) \tag{17}$$

and associate the local score $s_i(S)$ as the weight of this soft clause by defining

$$c((\neg P_i^S)) = s_i(S). \tag{18}$$

3.3 Summary of the Encoding

Assume an arbitrary instance $(X, W, \{s_i\}_{i=1}^N)$ of BTW-BNSL, consisting of a set $X = \{X_1, \ldots, X_N\}$ of nodes, a treewidth bound W, and for each X_i a non-negative local score (cost) $s_i(P_i)$ for each $P_i \in \mathcal{P}_i$ with $|P_i| \leq W$. The weighted partial MaxSAT instance $F(X, W, \{s_i\}_{i=1}^N) = (F_h, F_s, c)$ consists of the hard clauses corresponding to Eqs. 4–16 and the soft clauses corresponding to Eq. 17 with weights assigned according to Eq. 18.

Given an arbitrary solution τ to $F(X, W, \{s_i\}_{i=1}^N)$, the choice of the parent set S for each node X_i is given by the Boolean variable P_i^S for which $\tau(P_i^S) = 1$. We denote by G_τ the DAG corresponding to this choice S of a parent set for each node X_i.

Theorem 1 *For any solution τ to $F(X, W, \{s_i\}_{i=1}^N) = (F_h, F_s, c)$, let G_τ be the DAG corresponding to τ. It holds that τ is an optimal solution to $F(X, W, \{s_i\}_{i=1}^N)$ if and only if G_τ is an optimal solution the BTW-BNSL instance $(X, W, \{s_i\}_{i=1}^N)$.*

Proof. (*sketch*) Eq. 4 ensures that for each node X_i, $\tau(P_i^S) = 1$ for exactly one parent set $S \in \mathcal{P}_i$, i.e., a single parent set for X_i is chosen. Eqs. 5–8 ensure that G_τ is a DAG. Eqs. 10–11 ensure that the M_{ij} variables with $\tau(M_{ij})$ correspond exactly to the moralization of G_τ. Eq. 12 ensures that any assignment to the ord_{ij} variables corresponds to the linear ordering ord over X for which i comes before j iff $\tau(ord_{ij}) = 1$. Eqs. 14–15 encode exactly the conditions for an edge to be present in the triangulation of G_τ under ord, and Eq. 16 enforces the edge-directions of the triangulation according to ord, corresponding exactly to the ordered graph (consisting of the edges (X_i, X_j) for which $\tau(O_{ij}) = 1$) of G_τ under ord. Eq. 13 is satisfied iff there is a linear ordering ord, i.e., an assignment over the variables ord_{ij}, such that the maximum number of successors in the ordered graph represented by the O_{ij} variables is at most W. Finally, Eqs. 17–18 encode exactly the objective function of BTW-BNSL. $\square$

4 EXPERIMENTS

We present results on the efficiency of optimally solving the BTW-BNSL problem via our MaxSAT encoding using a state-of-the-art MaxSAT solver. As the MaxSAT solver we used MaxHS (Davies and Bacchus, 2013)[4]. For comparing to the recent exact approach to BTW-BNSL based on dynamic programming, we used the `best-w-tree` implementation available from the authors at `http://www.cs.helsinki.fi/u/jazkorho/aistats-2013/`.

The experiments were performed on a cluster of 2.8-GHz Intel Xeon quad core machines with 32-GB memory and Ubuntu Linux 10.04. A timeout of 8 h (28 800 seconds) and a memory limit of 30 GB were enforced on the solvers on the individual benchmark instances.

As benchmark data, we used a set of well-known UCI dataset with 9–29 variables. We used the MDL scoring function (Lam and Bacchus, 1994) for computing the local scores of parent sets from the datasets. Furthermore, we included as benchmarks the two datasets (Adult, Housing) made available by Korhonen and Parviainen (2013) with pre-computed local scores, giving a total of 10 datasets. As treewidth bounds, we used the values $W = 2, 3, 4$, resulting in a total of 30 benchmark instances. We pruned candidate parent sets using the following well-known pruning rule that maintains the set of optimal solutions: Given two parent sets $S, S' \in \mathcal{P}_i$, if $S' \subset S$ and $s_i(S') \leq s_i(S)$, then S can be pruned away from consideration. We observed that applying this pruning rule had a positive effect on the running times of both the MaxSAT solver and the dynamic programming approach. The pruning of a particular candidate parent set $S \in \mathcal{P}_i$ is reflected in the MaxSAT encoding by the fact that the corresponding Boolean variable P_i^S is not introduced.

Results are presented in Table 2 under treewidth bounds $W = 2, 3, 4$. For each bound, the best running time to find an optimal solution is highlighted in boldface.

We observe that the dynamic programming approach (**DP**) is competitive with our MaxSAT-approach only for the smallest dataset with 9 variables. Apart from the multiple timeouts ("> 28 800"), we observe that **DP** most often runs out of memory ("mo") on the datasets with more variables, especially for treewidth bounds greater than 2; memoryouts can be considered more critical than timeouts since they imply that the algorithm cannot give a solution however much time it is given. In contrast, the MaxSAT-approach (**MS**) timeouts on only two instances, and, especially, does not suffer from memouts. For a clear 2/3 majority of the instances, **MS** produces an optimal solution within half-an-hour; and for half of the instances within around 10 minutes.

5 RELATED WORK

Cussens (2008) formulated BNSL *without* treewidth restrictions as MaxSAT. Our encoding is more involved: we enforce a strict treewidth bound, and apply a more intricate encoding of the acyclicity constraint. Cussens used at-the-time state-of-the-art *local search* MaxSAT solvers, and was hence unable to find optimal networks, and also used integer-rounded local scores for candidate parent sets; in contrast we use a current state-of-the-art *complete* MaxSAT solver which pro-

[4]The developers of MaxHS provided a version which allows for assigning real-values as costs on soft clauses.

Table 2: Running times in seconds of our MaxSAT-based approach (**MS**) and the dynamic programming (**DP**) approach (Korhonen and Parviainen, 2013) for different UCI datasets and treewidth bounds $W = 2, 3, 4$. Explanations: "mo" denotes a memory out; N denotes the number of variables (nodes); **#fails** denotes the number of times the memory or time limit was exceeded.

		treewidth ≤ 2		**treewidth** ≤ 3		**treewidth** ≤ 4		**#fails**	
Dataset	N	**MS** (s)	**DP** (s)	**MS** (s)	**DP** (s)	**MS** (s)	**DP** (s)	**MS**	**DP**
Abalone	9	64	**7**	166	**57**	**215**	536	0	0
Housing	14	**2 226**	6 927	**2 329**	> 28 800	**2 991**	mo	**0**	2
Wine	14	**27**	6 924	**22**	> 28 800	**171**	mo	**0**	2
Adult	15	**998**	> 28 800	**1 623**	> 28 800	**1 782**	mo	**0**	3
Voting	17	**22 909**	> 28 800	**26 419**	mo	> 28 800	mo	**1**	3
Zoo	17	**410**	> 28 800	**412**	mo	**105**	mo	**0**	3
Hepatitis	20	**315**	mo	**100**	mo	**1 164**	mo	**0**	3
Heart	23	**1 198**	mo	**2 186**	mo	**41**	mo	**0**	3
Horse	28	**192**	mo	> 28 800	mo	**544**	mo	**1**	3
Flag	29	**1 418**	mo	**11 148**	mo	**1 356**	mo	**0**	3
#fails:		**0**	7	**1**	9	**1**	9	**2**	25

vides *provably optimal* solutions, and use the actual (non-integer) local scores without rounding.

Korhonen and Parviainen (2013) proposed an exact algorithm for BTW-BNSL based on dynamic programming. Their algorithm is also to our best knowledge the only approach for learning guaranteed-optimal bounded treewidth Bayesian network structures. We provide in this paper an empirical comparison: our MaxSAT-based approach scales both to larger numbers of variables and larger treewidth bounds than the dynamic programming approach.

Elidan and Gould (2008) proposed a greedy search strategy for learning Bayesian networks under treewidth constraints. Their algorithm relies on a search operator which is guaranteed to increase the treewidth of the current solution by at most one. Their approximation algorithm is polynomial-time in the number of variables and treewidth. However, due to the local search strategy, no bounds on the quality of the learned network can be guaranteed.

Ordyniak and Szeider (2013) consider the problem of learning and optimal network structure given a super-structure of bounded treewidth, and show that this problem is fixed parameter tractable in the treewidth of the super-structure. The treewidth of the super-structure does not, in general, bound the treewidth of the network, and hence does not ensure efficient exact inference after learning the network.

Integer-linear programming (ILP) provides another constrained optimization approach to BNSL, as studied by Jaakkola et al. (2010); Studený et al. (2010); Cussens (2011); Bartlett and Cussens (2013).

Finally, algorithms for learning undirected graphical models, especially, classes of Markov networks (Malvestuto, 1991; Bach and Jordan, 2001; Karger and Srebro, 2001; Srebro, 2003; Narasimhan and Bilmes, 2004; Chechetka and Guestrin, 2007; Gogate et al., 2010; Szántai and Kovács, 2012; Kumar and Bach, 2013) which enable fast inference by e.g., bounding the treewidth of the underlying tree-decompositions (often referred to as *junction trees*) have been developed. To our understanding, none of these algorithms guarantee to learn globally optimal structures.

6 CONCLUSIONS

Exact inference in low-treewidth Bayesian networks is tractable, which motivates the development of practical approaches to learning bounded treewidth networks. However, few practical algorithms have been proposed for learning networks under treewidth constraints. In this paper, we presented an approach to learning bounded treewidth Bayesian network structures that is guaranteed to provide optimal structures. Our approach is based on encoding the structure learning problem as weighted partial Maximum satisfiability, and then using a state-of-the-art MaxSAT solver for solving the resulting MaxSAT instances, i.e., for finding optimal bounded treewidth Bayesian network structures. We showed that our non-trivial MaxSAT encoding results in notably better performance compared to an implementation of a recently proposed dynamic programming algorithm for optimal bounded treewidth Bayesian network structure learning.

Acknowledgements

Work supported by Academy of Finland (COIN Centre of Excellence in Computational Inference Research, grant #251170) and Finnish Funding Agency for Technology and Innovation (project D2I). The authors thank Jessica Davies for providing the MaxHS version used in the experiments.

References

Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In *Proc. CP*, volume 8124 of *LNCS*, pages 80–96. Springer, 2013.

Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

Francis Bach and Michael Jordan. Thin junction trees. In *Proc. NIPS*, pages 569–576. MIT Press, 2001.

Mark Bartlett and James Cussens. Advances in Bayesian network learning using integer programming. In *Proc. UAI*, pages 182–191. UAUI Press, 2013.

Hans L. Bodlaender. Discovering treewidth. In *Proc. SOFSEM*, volume 3381 of *LNCS*, pages 1–16. Springer, 2005.

Anton Chechetka and Carlos Guestrin. Efficient principled learning of thin junction trees. In *Proc. NIPS*, pages 273–280. MIT Press, 2007.

David Maxwell Chickering. Learning Bayesian networks is NP-complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

David Maxwell Chickering. Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, 2002.

Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393 – 405, 1990.

Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

James Cussens. Bayesian network learning by compiling to weighted MAX-SAT. In *Proc. UAI*, pages 105–112. AUAI Press, 2008.

James Cussens. Bayesian network learning with cutting planes. In *Proc. UAI*, pages 153–160. AUAI Press, 2011.

James Cussens, Mark Bartlett, Elinor M. Jones, and Nuala A. Sheehan. Maximum likelihood pedigree reconstruction using integer linear programming. *Genetic Epidemiology*, 37(1):69–83, 2013.

Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in Maxsat. In *Proc. SAT*, volume 7962 of *LNCS*, pages 166–181. Springer, 2013.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2): 41–85, 1999.

Gal Elidan and Stephen Gould. Learning bounded treewidth bayesian networks. *Journal of Machine Learning Research*, 9:2699–2731, 2008.

Vibhav Gogate, William Webb, and Pedro Domingos. Learning efficient Markov networks. In *Proc. NIPS*, pages 748–756. MIT Press, 2010.

David Heckerman. A tutorial on learning with Bayesian networks. In *Learning in Graphical Models*, volume 89 of *NATO ASI Series*, pages 301–354. Springer, 1998.

David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila. Learning Bayesian network structure using LP relaxations. In *Proc. AISTATS*, 2010.

Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. *AI Magazine*, 33(1):89–92, 2012.

David Karger and Nathan Srebro. Learning Markov networks: maximum bounded tree-width graphs. In *Proc. SODA*, pages 392–401. SIAM, 2001.

Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, pages 549–573, 2004.

Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Proc. AISTATS*, pages 370–378, 2013.

K. S. Sesh Kumar and Francis Bach. Convex relaxations for learning bounded-treewidth decomposable graphs. In *Proc. ICML*, pages 525–533, 2013.

Wai Lam and Fahiem Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 19, pages 613–631. IOS Press, 2009.

Francesco M. Malvestuto. Approximating discrete probability distributions with decomposable models. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5), 1991.

Mukund Narasimhan and Jeff Bilmes. PAC-learning bounded tree-width graphical models. In *Proc. UAI*, pages 410–417. AUAI Press, 2004.

Ilkka Niemelä. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence*, 53 (1-4):313–329, 2008.

Sebastian Ordyniak and Stefan Szeider. Parameterized complexity results for exact Bayesian network structure learning. *Journal of Artificial Intelligence Research*, 46:263–302, 2013.

Sascha Ott and Satoru Miyano. Finding optimal gene networks using biological constraints. *Genome Informatics*, 14:124–133, 2003.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers Inc., 1988.

Marko Samer and Helmut Veith. Encoding treewidth into SAT. In *Proc. SAT*, volume 5584 of *LNCS*, pages 45–50. Springer, 2009.

Tomi Silander and Petri Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proc. UAI*, pages 445–452. AUAI Press, 2006.

Tomi Silander, Teemu Roos, Petri Kontkanen, and Petri Myllymaki. Factorized normalized maximum likelihood criterion for learning Bayesian network structures. In *Proc. PGM*, pages 257–272, 2008.

Nathan Srebro. Maximum likelihood bounded treewidth Markov networks. *Artificial Intelligence*, 143 (1):123 – 138, 2003.

Milan Studený, Jirí Vomlel, and Raymond Hemmecke. A geometric view on learning bayesian network structures. *International Journal of Approximate Reasoning*, 51(5):573–586, 2010.

Tamás Szántai and Edith Kovács. Hypergraphs as a mean of discovering the dependence structure of a discrete multivariate probability distribution. *Annals of Operations Research*, 193(1), 2012.

Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proc. UAI*, pages 584–590. AUAI Press, 2005.

Changhe Yuan and Brandon Malone. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 48: 23–65, 2013.

A-2014-3 L. Langohr: Methods for Finding Interesting Nodes in Weighted Graphs. 70+54 pp. (Ph.D. Thesis)

A-2014-4 S. Bhattacharya: Continuous Context Inference on Mobile Platforms. 94+67 pp. (Ph.D. Thesis)

A-2014-5 E. Lagerspetz: Collaborative Mobile Energy Awareness. 60+46 pp. (Ph.D. Thesis)

A-2015-1 L. Wang: Content, Topology and Cooperation in In-network Caching. 190 pp. (Ph.D. Thesis)

A-2015-2 T. Niinimäki: Approximation Strategies for Structure Learning in Bayesian Networks. 64+93 pp. (Ph.D. Thesis)

A-2015-3 D. Kempa: Efficient Construction of Fundamental Data Structures in Large-Scale Text Indexing. 68+88 pp. (Ph.D. Thesis)

A-2015-4 K. Zhao: Understanding Urban Human Mobility for Network Applications. 62+46 pp. (Ph.D. Thesis)

A-2015-5 A. Laaksonen: Algorithms for Melody Search and Transcription. 36+54 pp. (Ph.D. Thesis)

A-2015-6 Y. Ding: Collaborative Traffic Offloading for Mobile Systems. 223 pp. (Ph.D. Thesis)

A-2015-7 F. Fagerholm: Software Developer Experience: Case Studies in Lean-Agile and Open Source Environments. 118+68 pp. (Ph.D. Thesis)

A-2016-1 T. Ahonen: Cover Song Identification using Compression-based Distance Measures. 122+25 pp. (Ph.D. Thesis)

A-2016-2 O. Gross: World Associations as a Language Model for Generative and Creative Tasks. 60+10+54 pp. (Ph.D. Thesis)

A-2016-3 J. Määttä: Model Selection Methods for Linear Regression and Phylogenetic Reconstruction. 44+73 pp. (Ph.D. Thesis)

A-2016-4 J. Toivanen: Methods and Models in Linguistic and Musical Computational Creativity. 56+8+79 pp. (Ph.D. Thesis)

A-2016-5 K. Athukorala: Information Search as Adaptive Interaction. 122 pp. (Ph.D. Thesis)

A-2016-6 J.-K. Kangas: Combinatorial Algorithms with Applications in Learning Graphical Models. 66+90 pp. (Ph.D. Thesis)

A-2017-1 Y. Zon: On Model Selection for Bayesian Networks and Sparse Logistic Regression. 58.61 pp. (Ph.D. Thesis)

A-2017-2 Y. Hsieh: Exploring Hand-Based Haptic Interfaces for Mobile Interaction Design. 79+120 pp. (Ph.D. Thesis)

A-2017-3 D. Valenzuela: Algorithms and Data Structures for Sequence Analysis in the Pan-Genomic Era. 74+78 pp. (Ph.D. Thesis)

A-2017-4 A. Hellas: Retention in Introductory Programming. 68+88 pp. (Ph.D. Thesis)

A-2017-5 M. Du: Natural Language Processing System for Business Intelligence. 78+72 pp. (Ph.D. Thesis)

A-2017-6 A. Kuosmanen: Third-Generation RNA-Sequencing Analysis: Graph Alignment and Transcript Assembly with Long Reads. 64+69 pp. (Ph.D. Thesis)

A-2018-1 M. Nelimarkka: Performative Hybrid Interaction: Understanding Planned Events across Collocated and Mediated Interaction Spheres. 64+82 pp. (Ph.D. Thesis)

A-2018-2 E. Peltonen: Crowdsensed Mobile Data Analytics. 100+91 pp. (Ph.D. Thesis)

A-2018-3 O. Barral: Implicit Interaction with Textual Information using Physiological Signals. 72+145 pp. (Ph.D. Thesis)

A-2018-4 I. Kosunen: Exploring the Dynamics of the Biocybernetic Loop in Physiological Computing. 91+161 pp. (Ph.D. Thesis)